ABSTRACT

Reaching general conclusions through analysis of SourceForge data is difficult and error prone. Several factors conspire to produce data that is sparse, biased, masked, and ambiguous. We explore these factors and the negative effect that they had on the results of “Impact of Programming Language Fragmentation on Developer Productivity: a SourceForge Empirical Study.” In addition, we question the validity of evolutionary or temporal analysis of development practices based on this data.

1. INTRODUCTION

The present work began as a replication of a study by Krein, MacLean, Delorey, Knutson, and Eggert, “Impact of Programming Language Fragmentation on Developer Productivity: a SourceForge Empirical Study” [4]. This study explored the contributions of individual software developers in light of their use of multiple programming languages.

As authors of the original study, we desired to conduct a differentiated replication in order to explore the original question from another angle—that of project evolution. We hoped to clarify relationships between the nature of project growth and the language fragmentation of individual authors contributing to such projects. We expected that such a replication study would shed light on the potential impact of writing software in only one language, as opposed to developing in two or more languages.

In order to assess the impact of language fragmentation on project growth, we first needed to develop a technique for reliably measuring project growth in the context of our data set, which required understanding the growth patterns of projects on SourceForge. This intermediate objective yielded unexpected insights into the nature and usability of project data on SourceForge, particularly as it relates to the analysis of project evolution.

In this paper we present potential threats to validity, and insights into the limitations of SourceForge project data for understanding project evolution. In particular, we present our results in light of the replication we conducted to explore the effects on projects of developing software in multiple languages. In order to provide context for our insights, we provide background information on the original study which we sought to replicate. In addition to potential pitfalls inherent in SourceForge project data, we also describe threats to validity inherent in the study of language fragmentation and individual developer productivity using SourceForge data.

1.1 SourceForge as a Data Source

Over 100 researchers use the SourceForge Research Data Archive (SRDA)[5] hosted at Notre Dame to analyze development and distribution on SourceForge.net. Many utilize the data to study development behavior in open source projects.

Howison and Crowston enumerate several pitfalls in using SourceForge as a data source for research [2]. In section 2 we discuss our insights into the limitations of SourceForge for studying certain project attributes.

1.2 Language Fragmentation

The targeted study explored the correlation between language fragmentation and programmer productivity. Fragmentation is measured by language entropy (originally defined in [3]), with productivity defined as the number of lines of code committed to all “Production/Stable” or “Maintenance” phase projects on SourceForge in a given month. In order to provide context for our discussion of threats to validity, we briefly present the definitional premises of the original study.

1.3 Data Set

For this study and [4] the data set (Sample) comprises all projects designated “Production/Stable” or “Maintenance” as of the SRDA dump of SourceForge. Project history was gathered from project inception through October 2006. The data was originally collected for [1].

1.4 Definitions

In this study we define two terms precisely.

Definition 1 Daily Commit is the unit of work, defined as

1In addition to the intrinsic value of understanding open source development, some argue that OSS is sufficiently analogous to commercial software that global conclusions are justified.
the net contributions, in lines of code, for all authors to a project on a given day.

Although monthly contributions were the unit of work in the Fragmentation study, we found that Daily Commit unmasks certain attributes of the data that are not visible at a coarser granularity.

**Definition 2** Project Size is the total size of the project, in lines of code, at the most recent commit in the Sample.

## 2. PROJECT ATTRIBUTE PITFALLS

Several attributes of SourceForge projects may lead to erroneous results if not properly considered: 1) Unnatural Project Growth Patterns (which we refer to as Cliff Walls), 2) Auto-Generated Source Code, 3) Internal Development, 4) Development Pushes, and 5) Small Projects. To address the first four items we initially examine the Java eXPerience Framework (JXPFW) project as an example of problems in the data that lead to erroneous or artificially inflated results. After articulating the problems in the single case, we show that the same problems exist in a large percentage of the projects on SourceForge. Finally, we address the issues that arise when analyzing small projects.

### 2.1 Java eXPerience FrameWork

JXPFW is a moderately sized, “Production / Stable” project written primarily in Java. Other languages utilized in this project include XML, CSS, HTML, and XHTML. The project was chosen from 25 randomly selected projects because it had the largest Daily Commit.

As of August 6, 2006 JXPFW contained 160,946 total lines—placing it in the top quartile of all projects in the Sample—of which 63,720 were classified as source code\(^2\). At least 67,023 lines appear to be auto-generated files (discussed later).

### 2.2 Language Entropy

In order to explain the problems associated with using SourceForge data to analyze Language Fragmentation, we must first define entropy and its calculation. Entropy, defined as

\[
E(S) = -\sum_{i=1}^{c} (p_i \times \log_2 p_i) \tag{1}
\]

measures the “evenness” and “richness” of a distribution \((p)\) of classes \((c)\) in a system \((S)\). In Language Entropy “evenness” describes how evenly a developer contributes code in multiple languages. For example, if a developer committed 100 lines of Python and 100 lines of Java in one month, he or she would have maximize entropy for two languages: 1.0. For 150 lines of Python and 50 lines of Java Language Entropy would be 0.811. “Richness” describes the number of languages employed by an author in a given month. Maximum entropy is

\[
\log_2 c \tag{2}
\]

where \(c\) is the number of languages (classes).

The authors found a negative correlation between language entropy and programmer productivity. However, further analysis casts doubt on the generality of these conclusions.

### 2.3 Cliff Walls

Abnormal growth spikes in a project (Cliff Walls) constitute a serious threat to validity in evolutionary research utilizing SourceForge project data. These cliff walls represent periods of time during which data for the project is masked, missing, or ambiguous. Figure 1 shows the growth of the Java eXPerience FrameWork over time. The project was created on September 8, 1999 but sat dormant for over two and a half years before any source code was committed.

![Java eXPerience Framework Total Project Growth](image)

**Figure 1:** Growth of the Java eXPerience FrameWork over time.

On May 1, 2002, 138 new source code files were added to the project by a single author, totaling 18,675 lines of code (see Figure 2). In addition, another 62 lines were modified in 9 existing files. Auto-generated source code, internal development, gate-keepers, and development pushes are all developmental practices that could lead to these abnormally large commit sizes. We discuss each of these in turn in the following sections.

### 2.4 Auto-Generated Files

42% of the lines in JXPFW are auto-generated .mdl files marked as binary in CVS. However, unlike image files such as .gif, .mdl line count is included in the lines\_added statistic. In JXPFW these files are easy to identify due to their extreme size in proportion to the Project Size and the fact that they are marked binary. Unfortunately, in many projects auto-generated files are legitimate source code and don’t exhibit any telltale characteristics. For example, Java user interface code is often generated by tools rather than written by hand. These files can be difficult to identify but probably don’t represent a unit of work analogous to code written by hand.

No correction is made in the Language Fragmentation paper for auto-generated files, although the issue is listed in the threats to validity. These commits can significantly alter the result of the language entropy calculation. If auto-generated code is committed in a language that otherwise...
represents a small proportion of an author’s efforts (such as auto-generated XML configuration files), language entropy is artificially increased. Conversely, if the auto-generated code is in a dominant language (such as Java user interface code), language entropy is artificially decreased. Both results cast doubt on the validity of the original calculation.

2.5 Internal Development
Another possible cause of Cliff Walls is internal development not yet stored on SourceForge. Such activity may result from corporate sponsorship or co-located developers who find it easier to collaborate locally. In both cases, SourceForge essentially becomes a distribution tool rather than a collaboration environment. In fact, 12.2% of the projects were only active on a single day (1,221 of 9,997). In addition, 50% of the projects had fewer than 17 active days (5,004 of 9,997). One project, “ipfilter” was active for a two and a half hour period on August 6, 2006, in which 71,878 lines were checked in. No changes were made before the data was extracted four months later.

Additionally, projects may experience periods of internal development. For example, shortly before release commits may be restricted to only allow bug fixes. These intermittent stages of restrictive commits may cause periodic cliff walls.

When development occurs outside of SourceForge, the data committed to the public repository is of such coarse granularity that conclusions about development efforts and practices based on the revision data are suspect. In Figure 2 we see that there appears to be no development activity for the first two and a half years of the project. However, given the large commit size on May 1, 2002, it is probable that the growth approximates Figure 3. Unfortunately, without further data we can only make educated guesses.

2.6 Development Pushes
Although auto-generated files and internal development may be the culprits in some cases, in other cases large commits may simply indicate an impending deadline or “development push.” While it is unlikely that all of the developers on a project wrote 20% of a project in a single weekend, it is not impossible, and therefore cannot be discounted. Distinguishing between development pushes and artificially inflated commits is extremely difficult, and requires the acquisition of knowledge about a project from non-code sources such as email lists, bug reports, and interviews.

2.7 Generalizing Pitfalls
The Cliff Walls demonstrated in JXPFW occur frequently in the Sample. Over 4,000 projects are made up almost entirely of initial commits, meaning that the files were checked in at their maximum size (see Figure 4). This effect is most pronounced in projects whose size lies in the first quartile, and is only slightly less pronounced in the other three (see Figure 5).

2.8 Small Projects
The simplest explanation for cliff walls in the data is small project size. If a 4,000 line Daily Commit is made to a project with a size of 8,000 lines, that commit represents 50% of the Project Size. However, the same Daily Com-
mit to a project with a Project Size of 500,000 lines is negligible. Given the analytical impact of this phenomenon, we explore a possible explanation for small projects in the Sample, given the requirement that the projects are marked “Production/Stable” or “Maintenance.”

78% (3,197 of 4,094) of projects with Project Size less than 10,000 and 57% (5,688 of 9,997) of all projects in our Sample have only a single author. Projects that can be developed and maintained by a single author are generally smaller than those developed and maintained by a dozen or more authors. To complicate matters, a single author has no need for collaborative tools, and may therefore be less likely to “commit early, commit often.” Instead, small projects often exhibit peculiarities unique to an author and not relevant in discussions of collaborative product development.

3. AUTHOR BEHAVIOR PITFALLS

In addition to the pitfalls of project data that we discuss in the previous section, we observe a number of problems with author data that render analysis difficult or problematic. Several limitations are identified in the original study that we set about to replicate [4]. In light of the Cliff Walls and other limitations with projects, in this section we address Marginally Active Developers and Non-Contributing Months. Finally, we explore Author Project Size Bridging as an additional limitation.

3.1 Marginally Active Developers

Krein, et. al. [4] state that marginally active developers—those who contribute code during a limited number of months—may bias the results since they may be less likely to write in multiple languages. Observed at a granularity of one month, these authors represented few data points and therefore were less likely to generate realistic regression lines in the random coefficients model. We suggest addressing potential errors in analysis (discussed in Section 4) as well as using a finer granularity to mitigate the effects of low productivity. If developers are truly developing in multiple languages con-

currently, this approach would reveal that limitation while still capturing language entropy.

3.2 Non-Contributing Months

In addition to marginal activity, many developers don’t contribute regularly. Krein, et. al., recognize the potential problems with this data masking, and filter the data to remove abnormally large commits. However, given the Cliff Walls exposed in Section 2, this approach is likely insufficient to mitigate the potentially induced error.

Figure 3 shows a time period of two and a half years during which development occurred, but for which data is entirely missing. In [4] the entire development period would have been analyzed as the contributions for a single author (named keess) in the month of May, 2002. Classes for language entropy for the month include HTML, Java, XML, SQL, and CSS. Although this data point for JXPFW would have been excluded in [4] due to its size (19,881 lines of code), it is easy to imagine a similarly drawn-out development process with a smaller total size that fell below the threshold of exclusion. Alternatively, if four developers had collaborated on the JXPFW commit, it would not have been filtered.

Regardless of the size of Cliff Walls following months of inactivity, the potential for errors in the language entropy calculation is high. Researchers must take care to filter or categorize anomalous commits to accurately analyze developer activity.

3.3 Author Project Size Bridging

Analysis of author contributions reveals that authors don’t often bridge between project sizes (see Table 1). Authors tend to contribute to projects of a similar size and tend not to cross project size boundaries. To illustrate, we first must discretize projects into groups. Figure 6 shows a discretization by quartile on contributions ordered by project size. In other words, 25% of the contributions were to projects of size 0 to 102,852, 25% to projects of size 102,852 to 337,858, etc.

Figure 6: Project size groups.

\[ \text{Because } y \text{ is on the log scale, the bins towards the top of the histogram cover much greater range than the bins towards the bottom (notice the range of Group 1, 102,852, compared to the range of Group 4, 26,196,412). To elucidate this, we provide a boxplot. The boxplot demonstrates that most of the data in Group 4 are outliers.} \]
If projects are discretized into groups using these quartiles, only 6.82% (1,499 of 22,095) of authors contribute to projects in multiple groups. 93.18% of authors contribute within a single contribution group. Another 3.66% bridge only between contiguous groups. This general lack of bridging suggests that the author data represent distinct populations, rather than one. If true, this assertion requires that researchers block analysis of author productivity and contribution by contribution group.

When the same analysis is performed on only those authors who contribute to multiple projects, the contrast is not as extreme (see Table 2). However, only 13.14% (2,891 of 21,990) contribute to more than one project. Since removing single project authors also strips out nearly 90% of the data, doing so is not a viable solution.

Further analysis of author contribution is required to determine the meaning of the 5.15% of authors who bridge between contiguous groups. We expect that shifting the boundaries of the groups will increase the number of authors who contribute to a single group.

### 4. LIMITATIONS IN THE ORIGINAL STUDY

In [4], the authors use lines\textunderscore added as the metric for productivity. They argue that lines\textunderscore added captures developer productivity in two ways. First, new lines committed to a project are recorded in the metric. Second, a line modified is recorded as a lines\textunderscore added and lines\textunderscore removed. While this metric captures development after a file has been created, it misses a critical fact: a file has a size when it is committed. This size represents developer adjustments, SourceForge data is a fertile source from which researchers can draw conclusions about open source development. However, these conclusions must be tempered with proper awareness and appropriate methodological adjustments, SourceForge data is a fertile source from which to draw information and conclusions about open source development. However, these conclusions must be tempered with proper awareness and appropriate methodological adjustments.

<table>
<thead>
<tr>
<th>Group</th>
<th>Authors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11,632</td>
<td>52.90%</td>
</tr>
<tr>
<td>2</td>
<td>4,202</td>
<td>19.11%</td>
</tr>
<tr>
<td>3</td>
<td>3,024</td>
<td>13.75%</td>
</tr>
<tr>
<td>4</td>
<td>1,633</td>
<td>7.43%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Authors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2</td>
<td>633</td>
<td>2.88%</td>
</tr>
<tr>
<td>2, 3</td>
<td>133</td>
<td>0.51%</td>
</tr>
<tr>
<td>3, 4</td>
<td>58</td>
<td>0.26%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Authors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 3</td>
<td>340</td>
<td>1.55%</td>
</tr>
<tr>
<td>2, 4</td>
<td>151</td>
<td>0.69%</td>
</tr>
<tr>
<td>1, 4</td>
<td>66</td>
<td>0.30%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Authors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3</td>
<td>79</td>
<td>0.36%</td>
</tr>
<tr>
<td>2, 3, 4</td>
<td>8</td>
<td>0.04%</td>
</tr>
<tr>
<td>1, 2, 4</td>
<td>25</td>
<td>0.11%</td>
</tr>
<tr>
<td>1, 3, 4</td>
<td>17</td>
<td>0.08%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Authors</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3, 4</td>
<td>9</td>
<td>0.04%</td>
</tr>
</tbody>
</table>

**Table 1: Author bridging, or lack thereof, between Project Size groups (see Section 3.3).**

### 5. INSIGHTS AND CONCLUSIONS

While we’ve tried to articulate a series of caveats with respect to the use of SourceForge data for evolutionary analysis, we don’t intend to send an overly negative message. With proper awareness and appropriate methodological adjustments, SourceForge data is a fertile source from which to draw information and conclusions about open source development. However, these conclusions must be tempered with taxonomic caveats based on a full understanding of the problems we’ve identified in this paper.

#### 5.1 Mitigation of Project Problems

Unnatural project growth occurs in a high percentage of projects, depending on the definition of “unnatural.” Several questions must be answered in order to form a workable understanding:

1. What is the threshold of Daily Commit beyond which we can comfortably conclude that the data is not fully representative of the development effort?
2. How should that threshold of Daily Commit change based upon the number of authors contributing at a particular point in the project life cycle?
3. Is there a model that will expose, with high probability, projects for which the data is of sufficiently fine granularity that researchers can draw conclusions about
developmental and collaborative practices without requiring heavy qualification of the results due to data sparseness?

Answering these questions will provide researchers far more confidence in their results than is currently advisable.

5.2 Mitigation of Author Problems

Author problems may be slightly easier to overcome than project problems. Unlike project data, author data is derived from a single source, although there is some question whether or not source code always represents a single developer. Temporal analysis of a developer’s activities readily reveals unusual spikes in development, such as those at the beginning of Figure 7. After the initial spike, it appears that ‘keess’ has a somewhat normal commit pattern which could be used for analysis. However, further work is required to ensure that this assertion is correct.

![Figure 7: Development behavior of ‘keess.’](image)

5.3 Analytic Adaptations

SourceForge provides a wealth of data that, like other data sources, can easily be misinterpreted due to biases, masking, ambiguity, and sparseness. As researchers, by identifying pitfalls in the data and methods of compensation we increase the applicability of our results. These methods of analysis fortify our results against data irregularities and validate our exploration in this realm of open source software development.

5.4 Impact on the Original Study

The original study likely suffers from inflated language entropy numbers due to the cliff walls discussed herein as well as the exclusion of the initial size values. As discussed in Section 4, these omissions probably bias the study towards later stages of development when incremental changes are more prevalent than new source files. Ultimately, the study needs to be reevaluated in light of the findings discussed in this work.

5.5 Differentiated Replication

This study highlights the benefits of differentiated replication. The authors of the original study were satisfied that their work accurately summarized author programming language usage in SourceForge. However, when analyzed from a different angle—project development rather than single developer activity—it is apparent that they failed to account for several anomalies in the data. Although the authors in the original study strove to provide an unbiased, complete analysis of the data, the domain is simply too large to understand through a single study. Replication affords researchers new avenues and veins of exploration in partially explored areas and is a valuable tool to broaden and deepen understanding in a domain.

6. REFERENCES


