Finding the Core Developers

Jaco Geldenhuys
Computer Science Division, Department of Mathematical Sciences
Stellenbosch University, Stellenbosch, South Africa
Email: jaco@cs.sun.ac.za

Abstract—It has been suggested that 20% of the participants in a free/libre/open source software (FLOSS) project contribute 80% of the work. This paper attempts to verify this claim for nine projects and for various metrics of user activity such as the number of contributions, duration of involvement with the project, and the number of modifications to source code files.

Keywords-open source software, mining software repositories, developer behaviour

I. INTRODUCTION

Participants in free/libre/open source software projects (FLOSS) are often classified as core developers, peripheral developers and users—the so-called “onion model” [10]. Sometimes a finer distinction is drawn between peripheral developers who report bugs, and those who fix them. For some projects, usually large ones, it is also possible to identify leaders who act as managers and guide the overall direction of the project.

The 20/80 rule, first reported in the context of FLOSS projects by Mockus et al. [8], [9]. They observed that the top 15 developers contributed 83% of the commits and 88% of the lines of code in the Apache project. They then hypothesized that:

Open source developments will have a core of developers who control the code base, and will create approximately 80% or more of the new functionality. If this core group uses only informal, ad hoc means of coordinating their work, it will be no larger than 10-15 people.

They also suggested that for larger projects, more formal mechanisms (such as code ownership, structured software processes, or code inspections) are needed to coordinate development. Data from the Mozilla project (with a core team of 22 to 35) seem to support this. Dinh-Trong and Biemann tried to confirm this hypothesis for FreeBSD [3] and found that the number of participants who contributed 80% ranged from 28 to 42. Koch and Schneider report a similar trend for the GNOME project [7].

In a larger study, Singh et al. looked at 205 projects and found that, on average, 20% of the top developers contributed about 81% of the source code [11]. While they claim that this agrees with the studies mentioned above, this is not strictly true. Core developers comprised just 3.9% of the overall number of developers in the Apache project, 4.5–7.2% in the Mozilla project, 13.9–16.8% in the FreeBSD project, and 17.3% in the GNOME project. This does not mean that the 20/80 finding is inaccurate. Ghosh and Prakash report almost exactly the same figure: in their study of 3149 projects, 20% of the top contributors were responsible for 81.2% of contributions to the code base [4].

This raises interesting questions: why is it that the 80% cutoff values in the two large-scale studies are so different from those in the single-project studies? Why do the values differ so much from project to project? Is this a sensible definition of “core developer”? To address these questions, this paper explores a variety of metrics. Each metric measures some aspect of developer behaviour (such as the number of contributions), and the users are ranked according to the metrics. We then compute the correlation between these rankings to determine whether or not previous definitions of the term “core developer” is robust, in the sense that it is always clear who these core developers are.

II. DATA AND METHODOLOGY

Nine FLOSS projects are examined in this work: autoconf is an extensible package of macros that produce shell scripts to automatically configure software source code packages; coreutils is a collection of UNIX command-line utilities; linux is the Linux kernel release 2.6; mesa implements the OpenGL graphics standard; pciutils is a library for portable access to PCI bus configuration registers and utilities based on the library; samba is a free SMB and CIFS client and server for sharing files and printers; vlc is cross-platform multimedia framework, player and server for a variety of formats; wine implements the Microsoft Windows API and runs Windows programs under UNIX; and xserver is the standard implementation of the traditional X11 server.

Table I shows the number of authors, number of commits (i.e., updates in the version control software), and the date of the first and last commit for each project. The projects are all long-running, widely used, and could be classified as successful, although this was not a deciding factor in their selection, but is clearly related to their longevity. The projects represent a spectrum in terms of their size, number of developers, and application domain.
are classified into six categories: certain parts of a project. To investigate this possibility, files 7909 identifiers; 23.5% of them use more than one. Overall, 5605 individuals use 16 different identifiers. Nine users use eight or more different identifiers, and one user uses 99 identifiers. The identifiers in all the projects were manually compared and reconciled; this indicates that they agree on the identity of the core developers. The choice of metrics is somewhat subjective, but if metrics $\mu_1$ and $\mu_2$ are believed to both describe source code, testing code executed during development or installation, or even code used for the build process. In the end, many of the files were inspected manually. There are, no doubt, some misclassifications, and some files defy classification, but such cases are rare enough not to influence the results.

### Table I

<table>
<thead>
<tr>
<th>Project</th>
<th>Authors</th>
<th>Commits</th>
<th>First</th>
<th>git</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoconf</td>
<td>33</td>
<td>950</td>
<td>Mar 92</td>
<td>Sep 07</td>
<td>Mar 10</td>
</tr>
<tr>
<td>coreutils</td>
<td>73</td>
<td>2304</td>
<td>Oct 92</td>
<td>Oct 06</td>
<td>Mar 10</td>
</tr>
<tr>
<td>linux</td>
<td>3668</td>
<td>95949</td>
<td>Apr 02</td>
<td>Apr 05</td>
<td>May 08</td>
</tr>
<tr>
<td>mesa</td>
<td>243</td>
<td>23397</td>
<td>Feb 98</td>
<td>Dec 06</td>
<td>Mar 10</td>
</tr>
<tr>
<td>pcpuilts</td>
<td>15</td>
<td>317</td>
<td>Dec 97</td>
<td>May 06</td>
<td>Jan 10</td>
</tr>
<tr>
<td>samba</td>
<td>78</td>
<td>5950</td>
<td>May 96</td>
<td>Aug 07</td>
<td>Sep 09</td>
</tr>
<tr>
<td>vlc</td>
<td>216</td>
<td>15491</td>
<td>Aug 99</td>
<td>Mar 08</td>
<td>Mar 10</td>
</tr>
<tr>
<td>wine</td>
<td>554</td>
<td>43152</td>
<td>Jun 93</td>
<td>Nov 05</td>
<td>Mar 10</td>
</tr>
<tr>
<td>xserver</td>
<td>252</td>
<td>6874</td>
<td>Nov 99</td>
<td>Jun 06</td>
<td>Mar 10</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Rank</th>
<th>A</th>
<th>C</th>
<th>L</th>
<th>M</th>
<th>P</th>
<th>S</th>
<th>V</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>55.4</td>
<td>72.1</td>
<td>4.1</td>
<td>17.1</td>
<td>87.7</td>
<td>25.4</td>
<td>26.2</td>
<td>11.4</td>
<td>21.8</td>
</tr>
<tr>
<td>2</td>
<td>21.7</td>
<td>6.7</td>
<td>1.7</td>
<td>13.6</td>
<td>5.0</td>
<td>20.3</td>
<td>12.3</td>
<td>6.0</td>
<td>8.9</td>
</tr>
<tr>
<td>3</td>
<td>9.2</td>
<td>5.8</td>
<td>1.7</td>
<td>8.3</td>
<td>2.2</td>
<td>16.1</td>
<td>11.9</td>
<td>5.8</td>
<td>8.0</td>
</tr>
<tr>
<td>4</td>
<td>3.7</td>
<td>3.6</td>
<td>1.7</td>
<td>6.5</td>
<td>1.6</td>
<td>13.3</td>
<td>8.3</td>
<td>4.7</td>
<td>7.5</td>
</tr>
<tr>
<td>5</td>
<td>2.0</td>
<td>1.6</td>
<td>1.3</td>
<td>4.4</td>
<td>0.3</td>
<td>7.4</td>
<td>8.1</td>
<td>4.6</td>
<td>7.0</td>
</tr>
</tbody>
</table>

### A. Identifying the Users

A first step in classifying user roles is identifying the users. This can be complicated, because the same user may be identified by different names. For example, although “John Smith” may be a common name, it is reasonable to assume that the following names all refer to the same person:

1. j.smith@y.z John Smith
2. j.smith@x.y.z John Smith
3. j.smith@y.z John Q. Smith
4. bunny@w.v John Q. Smith

The term identifier is used to refer to the combination of email address and real name. The identifiers in the nine projects were manually compared and reconciled: 36 users use eight or more different identifiers, and one user uses 16 different identifiers. Overall, 5605 individuals use 7909 identifiers; 39.5% of them use more than one.

### B. Project composition

One possible behaviour is that “core” developers focus on certain parts of a project. To investigate this possibility, files were classified into six categories:

1) MAK: controlling the build process (e.g., Makefiles);
2) CNF: for configuration of installed software;
3) SRC: program code, either compiled or interpreted once installed;
4) TST: source code and files associated with testing;
5) DOC: (user and developer) documentation; and
6) LOG: development history and notes.

This classification can be automated, but only partially. For instance, a file called test_input.c might contain shell scripts, standard Unix tools (e.g., awk, sort), and gnuplot were used to generate the figures and tables in this paper. The scripts and their output are available at http://www.cs.sun.ac.za/~jaco/CORE/.

### III. Rearranging the 20/80 Experiment

The first part of Table II shows the five participants who produced most of the commits (each project is identified by the first letter of its name). The values are percentages of the total number of commits. For instance, in the wine project, the top developer contributed 11.4% of the commits. The bottom two lines shows the percentage and actual number of participants who produced 80% of the commits.

It is clear that the projects differ significantly. Smaller projects (autoconf, coreutils, pcpuilts) appear much less egalitarian than the larger ones (linux, wine). The top contributors fall far short of the predicted 20%, ranging instead from 3.1% to 8.9%. These values agree with the single-project studies, but the actual number of developers ranges from very small (1), to surprisingly large (327), and clearly contradicts the values reported in the literature, also those in the single-project studies. The values do not seem to be correlated with project size.

Accurate identification of users does not explain these results; if the table is computed based purely on email addresses, the results are more or less the same. Bias in the choice of projects may also skew the cutoff values, but if, for example, the age or success of a project impacts the number of top developers, the expected bias is towards either a lower or higher number of core developers. Instead, some values are higher and others lower than the 10–15 bound proposed by Mockus et al.

### IV. User Ranking

Below, a set of metrics is defined and justified. The aim is to capture different aspects of user behaviour. For each metric, the users are ranked 1, 2, 3, …and the correlation between the metrics is computed. If a pair of metrics is consistently highly correlated across all of the projects, this indicates that they agree on the identity of the core developers. The choice of metrics is somewhat subjective, but if metrics $\mu_1$ and $\mu_2$ are believed to both describe...
A. Metrics

Consider a sequence of commits $c_1, c_2, \ldots, c_n$ made by user $u$. Let $t(c)$ be the time of commit $c$.

1) Commit Count, Timespan, and Frequency: The most straightforward metrics are the number of commits $\alpha_1(u) = n$, the commit timespan $\alpha_2(u) = t(c_n) - t(c_1)$, and the commit frequency $\alpha_3(u) = n/\alpha_2(u)$. The intuition is that core developers are likely to commit more times, more often, and over a longer period of time.

2) File Modifications: For a commit $c$, define $m(c)$ as the number of files modified in $c$. Define as metrics the size of the largest commit $\beta_1(u) = \max_i m(c_i)$, the total number of files modified $\beta_2(u) = \Sigma_i m(c_i)$, and the average number of files modified per commit $\beta_3(u) = \beta_2(u)/n$. Justification: core developers may be called on to make large commits that modify many files, they may modify more files during their involvement with a project, and may—on average—modify more files in each commit than other users.

3) Lines Added and Deleted: For a commit $c$, define $\ell_a(c)$ as the number of lines added in commit $c$, and $\ell_d(c)$ as the number of lines deleted. Define as metrics the total number of lines added $\gamma_1(u) = \Sigma_i \ell_a(c_i)$, and the total number of lines deleted $\gamma_2(u) = \Sigma_i \ell_d(c_i)$. The number of commits alone does not accurately reflect the effort of contribution; a participant could, in theory, make only a few commits, but add many new lines to the code base.

4) Files Added and Deleted: For a commit $c$, define $f_a(c)$ as the number of files added in commit $c$, and $f_d(c)$ as the number of files deleted. Define as metrics the total number of files added $\delta_1(u) = \Sigma_i f_a(c_i)$, and the total number of files deleted $\delta_2(u) = \Sigma_i f_d(c_i)$. It seems reasonable that newcomers may be reluctant to change a project’s structure in a major way, while more experienced (i.e., core) developers need to make structural changes to a project.

5) Class Modifications: For a commit $c$, define $C_x(c)$ as the number of files modified of class $x$, where $x \in \{M, C, S, T, D, L\}$ corresponds to MAK, CNF, SRC, TST, DOC, and LOG. Define as metrics the total number of class $x$ files modified, $\epsilon_x(u) = \Sigma_i C_x(c_i)$.

B. Ranking and correlations

The users were ranked according to each metric, with ties resolved consistently according to a unique user identifier. This introduces a linearity among equally-ranked users which boost the correlation, but which also makes it more robust against outliers.

The pairwise (Pearson’s product-moment) correlation between rankings was computed for all of the projects. Note that the correlation coefficients are perfectly significant for each project; correlation was calculated over all the users and no sampling was used. Sample scatterplots are shown in Figure 1. The dotted lines indicate the elusive 80% cutoff point for the metrics. In other words, in the $\alpha_1-\alpha_2$ scatterplot for the $xserver$ project, points to the right of the vertical dotted line represent the top-ranked users who contributed 80% of $\alpha_2$, i.e., whose commits accounted for 80% of the file modifications.

Table III shows the minimum (below the diagonal) and maximum (above the diagonal) correlations coefficient for each of the pair of metrics, taken over all of the projects. While some metrics agreed strongly for some projects, the surprising and important aspect of the table is the low minimum values. For at least one project, almost all the metrics disagreed on who the core developers are.

The strongest pairings are $\alpha_1/\beta_2$ (0.920…1.000), $\alpha_1/\alpha_2$ (0.918…1.000), and $\alpha_2/\beta_2$ (0.863…1.000). In retrospect, this is hardly surprising: the longer a developer is involved in a project ($\alpha_2$), the more commits he/she is likely to make ($\alpha_1$) and the more files he/she is likely to modify ($\beta_2$).

V. Discussion

The message of this paper is a negative one. None of the metrics proposed agreed strongly on who the core developers are, suggesting that the concept of “core developer” is not necessarily as well-defined as the literature would suggest. This casts doubt on the 20/80 rule and its variants. Of course, it is possible that more sophisticated metrics may cast a different light on the question. In general, more fine-grained analysis is required to determine the roles that developers undertake in FLOSS projects [1], [6].

An even more serious problem is that the nine projects investigated did not exhibit the same patterns of developer involvement. As many others have noted, FLOSS projects are not only significantly different from traditional, commercial projects, but they also differ among themselves. What is true for one project, may be false for another. And an even more subtle caveat is that care must be taken when making general statements, even about a single project. Since FLOSS projects do not, generally speaking, follow a formalized software process, it is difficult, if not impossible, to identify particular software lifecycle stages. As a project evolves, the behaviour of participants changes, and it is important to recognize that what is true for a particular project may change over time.

This poses somewhat of a problem, since researchers would like to understand FLOSS software processes and identify trends across different projects. Without such patterns, predicting aspects such as project success becomes more difficult, if not impossible.

REFERENCES


