Assessing the Representativeness of Open Source Projects in Empirical Software Engineering Studies

Hao Zhong and Ye Yang
Laboratory for Internet Software Technologies
Institute of Software, Chinese Academy of Sciences
Beijing, 100190, China
Email: {zhonghao,ye}@itechs.iscas.ac.cn

Jacky Keung
Department of Computing
The Hong Kong Polytechnic University
Kowloon, Hong Kong
Email: Jacky.Keung@comp.polyu.edu.hk

***Abstract*—*BACKGROUND:* Software engineering researchers have carried out many empirical studies on open source software (OSS) projects to understand the OSS phenomenon, and to develop better software engineering techniques. Many of these studies typically use only a few successful projects as study subjects. Recently, these studies have received criticisms and challenges on their representativeness on OSS projects.**

***AIM:* First, we aim to examine to what extent data extracted from successful projects are different from data extracted from the majority. If data extracted from successful projects are quite different from data extracted from the majority, approaches that are effective on successful projects may not be effective in general. Second, we aim to examine whether successful OSS projects are representative to the whole population of OSS. If they are not, conclusions that are drawn from only successful projects may reflect the OSS phenomenon partially.**

***METHODOLOGY:* We analyzed 11,684 OSS projects that are hosted on SourceForge. When researchers select subjects, they typically select successful projects that are attractive to both users and developers. Considering this preference, we clustered these projects into four categories based their attractiveness to users and developers. Here, we use the K-means clustering technique to produce combined result. Furthermore, we selected eight indicators that are used in many existing studies (*e.g.*, team sizes), and compared indicators that are extracted from different categories to investigate to what degree they are different.**

***RESULT:* For the first research aim, the result shows that 66.1% projects are under developing projects; 14.7% projects are user-preference projects; 14.2% projects are developer-preference projects; and only 5.0% projects are considered successful. For the second research aim, the result shows that all the eight analyzed indicators are highly unbalanced with the gamma distribution. Furthermore, the result reveals that users and developers of SourceForge have different perceptions on the development status defined by SourceForge.**

***CONCLUSION:* We conclude that successful projects are not representative to the whole population of OSS, and data extracted from successful projects are quite different from data extracted from the majority. The result implies that conclusions drawn from only a few successful projects may be challenged. This work is important to allow researchers to refine conclusions of existing studies, and to better understand and to carefully select OSS project subjects for their future empirical experiments.**

I. Introduction

Open source software (OSS) is a global phenomenon that attracts much attention from stakeholders with diverse backgrounds. Ever since its inception, many advocated developers and commercial organizations have put forward tremendous efforts to the OSS development, and benefit from knowledge exchange and be able to rapidly drive innovations [41]. With their contributions, open source portals such as SourceForge now host millions of OSS projects. Among these projects, especially those successful ones (*e.g.*, Linux) have attracted many users and developers.

The OSS phenomenon also attracts much attention from academic research. Up to the present, researchers have conducted many studies on OSS projects to understand the OSS phenomenon, or to evaluate software engineering approaches. Scacchi [33] surveyed existing studies, and point out that these studies cover various perspectives of OSS (see Section II for detailed descriptions of some such studies). Nevertheless, some researchers criticize that conclusions drawn in many existing studies may not be fully representative, since these studies typically select only few successful projects for analysis. In particular, Crowston *et al.* [11] reviewed more than one hundred published empirical studies on OSS projects. Their result shows that most studies selected fewer than ten successful projects for analysis (see Section II for the subject selection in some existing studies on OSS). Considering this undesirable trend, Crowston *et al.* [11] challenge the representativeness of existing empirical studies on OSS, and suspect that conclusions drawn from those successful OSS projects such as Linux might be significantly different from conclusions drawn from the majority of OSS projects (see the third paragraph of Section 3.3 in their paper [11] for details). Crowston *et al.*'s challenge shocked the foundation of existing studies on OSS, and needs to be responded with a more systematic empirical analysis. In particular, various researchers attempted empirical studies on the OSS projects to understand software engineering theories and practices (see Section II for details). For these studies, if data extracted from their selected subjects are quite different from data extracted from the majority of projects, the reliability of their conclusions could be challenged, and their theories or practices may not be truly reflected in general. Researchers also conducted various empirical studies on the OSS projects to understand the OSS phenomenon (see Section II for details). For these studies, if selected subjects are not the majority of the OSS projects, their drawn conclusions on OSS may not be representative,

and may only partially reflect the reality of OSS.

In response to Crowston *et al.*'s challenge, we attempted to answer some of many related questions. Are successful OSS projects the majority of the whole population of OSS projects? Are data extracted from successful OSS projects significantly different from data extracted from other projects? These questions are still open, and to answer these questions, there is a strong need for a quantitative study on the representativeness of OSS projects with regard to their success. However, such a quantitative study on the representativeness of OSS projects is challenging, since it requires a carefully designed analysis methodology and tremendous effort to analyze statistics from a large number of OSS projects.

In this paper, we propose a methodology that extracts statistics from OSS projects and analyzes extracted data to answer the preceding research questions. In literature, existing studies tend to select successful projects that are attractive to both users and developers. As a result, we define the success of an OSS project as its attractions to users and developers. Based on the definition, we conduct the first empirical study on individual indicators and the representativeness of OSS projects related to their project success. The result of this study allows researchers to refine their conclusions, and to select better subjects in their future studies. To the best of our knowledge, this work is first of its kinds to quantitatively analyze OSS projects.

This study makes the following contributions:

- A methodology that analyzes the representativeness of OSS projects concerning to their project success. Based on their attractions to users and developers, we use a clustering technique to build four clusters of projects: under developing projects, user-preference projects, developer-preference projects, and successful projects .
- The first empirical analysis on the representativeness of OSS projects concerning to their success. Our result shows that successful projects are the minority of OSS projects. To present a full overview of OSS, we suggest that researchers should analyze more user-preference projects, developer-preference projects, and under developing projects.
- An analysis on all the eight indicators used in our empirical study. We find that indicators of successful projects are typically thousand times larger than indicators of the majority, therefore conclusions drawn from successful projects can be significantly different from conclusions drawn from the majority. If researchers need to claim that their conclusions are general to all OSS projects, they should select subject projects from all the four categories.
- More analyzed result on OSS. For example, we find that existing channels fail to collect user feedbacks. As user feedbacks are important for OSS developers to improve their projects, we suggest that researchers should provide better channels to collect user feedbacks. As another example, we find that users and developers have different perceptions of some development status, and we suggest that development status should be more clearly defined to eliminate the difference.

Our study reveals some realities of OSS. For example, our result shows that OSS projects typically have small teams. However, our result does not provide any comparisons between open source software and closed source software, and we further discuss this issue in Section V. The rest of our paper is as follows. Section II presents related work. Section III presents our analysis methodology. Section IV presents our empirical study. Section V discusses issues of our approach. Section VI concludes this paper.

II. Related Work

This section briefly discusses the related important research to our empirical study.

Empirical studies to understand software engineering theories or practices. Researchers conducted various empirical studies on OSS projects to understand software engineering theories or practices. Capra *et al.* [8] conclude that when a project comes to an end, its governance becomes less formal, and its development effort increases. Gyimothy *et al.* [19] validate that some object-oriented metrics (*e.g.*, couplings between object classes) can be used to predict the fault-proneness of classes. Kim *et al.* [25] conclude that refactoring clones may not always improve software qualities. Kim *et al.* [24] also conclude that refactoring may introduce more defects, and the benefits of refactoring should be rethought. Shi *et al.* [35] enumerate various findings on the documentation evolution of five open source API libraries. Bachmann *et al.* [3] conclude that most committed bug reports are not related to bug fix or feature request. Bird *et al.* [4] conclude that biases exist in defect datasets, and such biases have negative impacts on defect prediction techniques. Aversano *et al.* [2] conclude that design patterns are suitable to support software that tends to change more frequently. Gabel and Su [14] conclude that there is a general lack of uniqueness in software. Ko *et al.* [26] conclude that word frequencies in bug reports generally follow Zipf's law. Harman and McMinn [20] enumerate various findings on search based testing. Grechanik *et al.* [17] enumerate various findings on code structures of Java programs. Pandita *et al.* [30] present the effectiveness of their approach with two popular APIs. Zhong *et al.* [43] conclude that some simple test-suite-reduction approaches are as effective as some complicated ones. Hindle *et al.* [21] reveal that code is even more repetitive than natural languages. Wang *et al.* [38] compare ten information retrieval techniques on their effectiveness of recovering the links between concerns and implementing functions. Schuler and Zeller [34] report that the dynamic slice of covered statements that influence an oracle is a good indicator for oracle quality. Polikarpova *et al.* [32] compare inferred contracts with programmer-written contracts. Zhang *et al.* [42] show that operator-based mutant selection is not superior than random mutant selection. These preceding studies typically analyze only several successful projects, and the representativeness of selected projects may affect their conclusions.

TABLE I
SUBJECT SELECTIONS AND USED DATA SOURCES IN SOME EXISTING STUDIES ON OSS

Author [Ref]	Subject	Data source
Mockus *et al.* [28]	Mozilla and Apache	email lists, cvs archives, and bug reports
Ye and Kishida [41]	GIMP	email lists and cvs archives
Paulson *et al.* [31]	Linux, Apache, and Gcc	code metrics (*e.g*, growing rates)
Huntley [22]	Apache and Mozilla	cvs archives
Dinh-Trong *et al.* [13]	FreeBSD	email lists, cvs archives, and bug reports
Kim *et al.* [25]	Carol and DNSJava	cvs archives
Gyimothy *et al.* [19]	Mozilla	code metrics (*e.g.*, methods per class)
Gurbani *et al.* [18]	a telephony server	team sizes, lines of code, numbers of downloads, and *etc.*
Bird *et al.* [5]	Apache, Ant, Python, Perl, and PostgreSQL	email lists
Sowe *et al.* [36]	Debian	email lists
Capra *et al.* [8]	75 projects	team sizes, lines of code, numbers of downloads, and *etc.*
Olbrich *et al.* [29]	Lucene and Xerce 2	cvs archives
Koch [27]	about 100 projects	numbers of downloads, web hits, team sizes, and *etc.*
Boulanger [6]	Linux, Apache, and MySQL	defect densities and *etc.*

Empirical studies to understand the OSS phenomenon. Researchers conducted various empirical studies on OSS projects to understand the OSS phenomenon. Mockus *et al.* [28] conclude that OSS projects overall can have lower defect densities and higher productivity than commercial software. Paulson *et al.* [31] conclude that OSS project does not have higher productivity, but defects in OSS projects are more rapidly found and fixed than commercial software. Ajila and Wu [1] conclude that software companies can achieve higher productivity and quality if they reuse OSS projects in a systematic way. Penta and German [15] conclude that explicit contributors and copyright owners are not necessarily the most frequent committers. Gurbani *et al.* [18] enumerate various lessons and open questions on the OSS development. Jensen and Scacchi [23] conclude that OSS projects have different ways to migrate their participants from peripheral roles to core leadership positions. Godfrey and Tu [16] conclude that Linux is growing faster than they expected. Bird *et al.* [5] conclude that OSS projects can have latent social networks. Sowe *et al.* [36] conclude that OSS developers communicate frequently, and their communication follows the fractal cubic distribution. Yamauchi *et al.* [40] conclude that the communication of OSS developers heavily relies on electronic media. Ye and Kishida [41] conclude that learning is one of the major driving forces that attract OSS developers. Dagenais and Robillard [12] conclude that the documentation development of OSS follows three production models: initial effort, incremental changes, and bursts. The preceding studies typically select only few successful OSS projects to analyze. Again, the preceding conclusions may provide many valuable insights on the OSS phenomenon, but may not be generalizable due to the selection of study subjects.

Characteristics of OSS projects. Capiluppi *et al.* [7] analyzed 406 projects hosted on Freshmeat for characteristics of OSS projects. As they analyzed data manually, only about four hundred projects were analyzed, and their data were extracted eight years ago. On the contrary, as we analyze statistics automatically, our empirical study analyzes much more up-to-date projects. Furthermore, our study focuses on the representativeness of OSS projects with regard to their success, instead of presenting general characteristics.

Subject selection in literature. Table I shows selected OSS projects and used data sources of some existing studies. Column "Author [Ref]" lists authors and references of these studies. In this table, we focus on studies that were published in the past ten years. Column "Subject" lists selected subjects of these studies. We find that except the two studies (*i.e.*, Capra *et al.* [8] and Koch [27]), all the other studies in Table I select only few successful OSS projects as subjects. Column "Data source" lists major data sources used in these studies. We find that many data sources are relevant to the indicators that are analyzed in our methodology. For example, Gurbani *et al.* [18], Capra *et al.* [8], and Koch [27] all use team sizes as data sources, and our empirical study shows that the indicator of team sizes is highly unbalanced. Besides direct usages, we find that some data sources used in existing studies rely on our indicators indirectly. For example, Mockus *et al.* [28], Ye and Kishida [41], Dinh-Trong *et al.* [13], Bird *et al.* [5], and Sowe *et al.* [36] all use email lists as data sources. Email lists indirectly rely on team sizes, and small teams typically have quite different email communications with large teams.

III. METHODOLOGY

Our methodology implements the following steps:

Step 1: Selecting OSS projects as subjects. The statistics from a single project can be random or even false. For example, a programmer may build tools to download released files from a project automatically to increase download numbers, since download numbers have impacts on user choices. To reduce the negative impacts from such false statistics, we select a large population of OSS projects as subjects, since most projects do not produce false statistics to deceive users. As we select thousands of projects for analysis, false statistics from few projects should have little negative influence to our conclusion. We define the success of an OSS project as its attractions to users and developers. For users and developers, different development status can have different attractions. In particular, SourceForge defines the seven development status: *inactive*, *mature*, *production/stable* (referred to as *production* for short), *beta*, *alpha*, *pre-alpha*, and *planning*. To ensure the representativeness of our study, we retrieve projects of all the seven development status for analysis.

Date (UTC)	Read Transactions	Write Transactions	Total Files Updated
04 Nov 2010 -	0	0	0
03 Nov 2010	0	0	0

Fig. 1. Some statistics on SourceForge

Step 2: Extracting statistics of subjects. Through the website of a project, SourceForge provides various statistics of the project. For example, Figure 1 shows an HTML file returned from SourceForge, and the file lists numbers of read transactions, numbers of write transactions, and numbers of total updated files on the two days (*i.e.*, 3rd and 4th, November, 2010). We implemented a tool that parses all these HTML files to extract statistics. A value of a single day can be random, and sums within a duration can better reflect the natures of a project. The duration should not be too short or too long, since sums within a too short duration cannot fully reflect the natures of a development status, and sums within a too long duration may reflect natures of multiple development status. Capiluppi *et al.* [7] state that 90% OSS projects do not change their development status over six months. For the preceding considerations, we choose to calculate sums within two months. Among extracted statistics, we choose four user-related indicators as follows to analyze user preference:

The sum of web hits. SourceForge provides web servers for its hosted projects. On these server, developers can release their websites, and users can browse their contents for projects of their interests. This indicator reflects the attraction of a project to users for browsing its contents.

The sum of file downloads. SourceForge provides file servers for its hosted projects. On these server, developers can release executable files of their projects, and users can download these files for a try. If a user feels that a project is indeed useful, the user may download its released files for a trial. This indicator reflects the attraction of a project to users for trying it.

The sum of forum posts. SourceForge provides forums for all its hosted projects. In these forums, developers and users can discuss various issues about their projects. The indicator reflects the attraction of a project to users for participating.

The sum of opened bugs. SourceForge provides bug-report servers to all its hosted projects. On these servers, users can report bugs, and developers can fix reported bugs. For example, the bug report server of Notepad++[1] lists all its bug reports, and most of them are from users. If a user finds a bug of a project, the user can report the bug to the project, so the indicator reflects the attraction of a project to users for improving its quality.

We choose the four developer-related indicators as follows to analyze developer preference:

Team sizes. In SourceForge, the website of a project lists names of all its developers. If a developer makes contributes continually to a project, the developer may require to be added to the developer list of the project. The indicator reflects the attraction of a project to developers for the identification of its team.

The sum of read transactions. For the ease of the collaboration among developers, SourceForge provides concurrent versions system (CVS) for its hosted projects. From CVS, developers can check out source files of projects. If a developer is interested in a project, the developer can check out its source files to learn. In other programming tasks such as code review, developers can also check out source files of OSS projects. As a result, the indicator reflects the attraction of a project to developers for reading its code.

The sum of write transactions. Through CVS of a project, some developers have the right to check-in source files. As developers need to write files after development, the indicator reflects the attraction of an OSS project to developers for collaborating.

The sum of updated files. When developers check in source files to a project, some files of the project may be updated. As each write transaction can update only one file or many files, the indicator reflects the attraction of an OSS project to developers for putting programming efforts on the project.

Step 3: Analyzing extracted statistics. Our study aims to answer the following two research question:

- RQ1: To what extent are indicators that are extracted from successful OSS projects different from indicators that are extracted from other OSS projects?
- RQ2: To what extent do successful OSS projects represent the whole population of OSS projects?

To answer the first research question, we present cumulative percents and various quantile-quantile plots (Q-Q plots) of all the indicators, since cumulative percents and Q-Q plots are both widely used to compare distributions of samples [39]. From the result, we are able to see the differences between data sources in successful projects and data sources in other projects, and estimate whether conclusions drawn from successful projects can be significantly different from conclusions drawn from the majority of OSS projects or not.

To answer the second research question, we cluster all the OSS projects into the following four categories:

(1) User-preference projects: These projects are attractive to users, but are not attractive to developers.

(2) Developer-preference projects: These projects are attractive to developers, but are not attractive to users.

(3) Successful projects: These projects are attractive to both users and developers, and are considered successful.

(4) Under developing projects: These projects are not attractive to either users or developers.

Values of different indicators are not comparable, and cannot be used for clustering directly. We first normalize all the extracted indicators to make them comparable. After normalization, for each project, we use the sum of user-related indicators to present its user preference, and use the sum of developer-related indicators to present its developer preference. Based on the preceding two sums, we use a clustering technique described in [37] to put projects into categories, and calculate percents of all the categories to present their representativeness.

[1]http://sourceforge.net/tracker/?group_id=95717&atid=612382

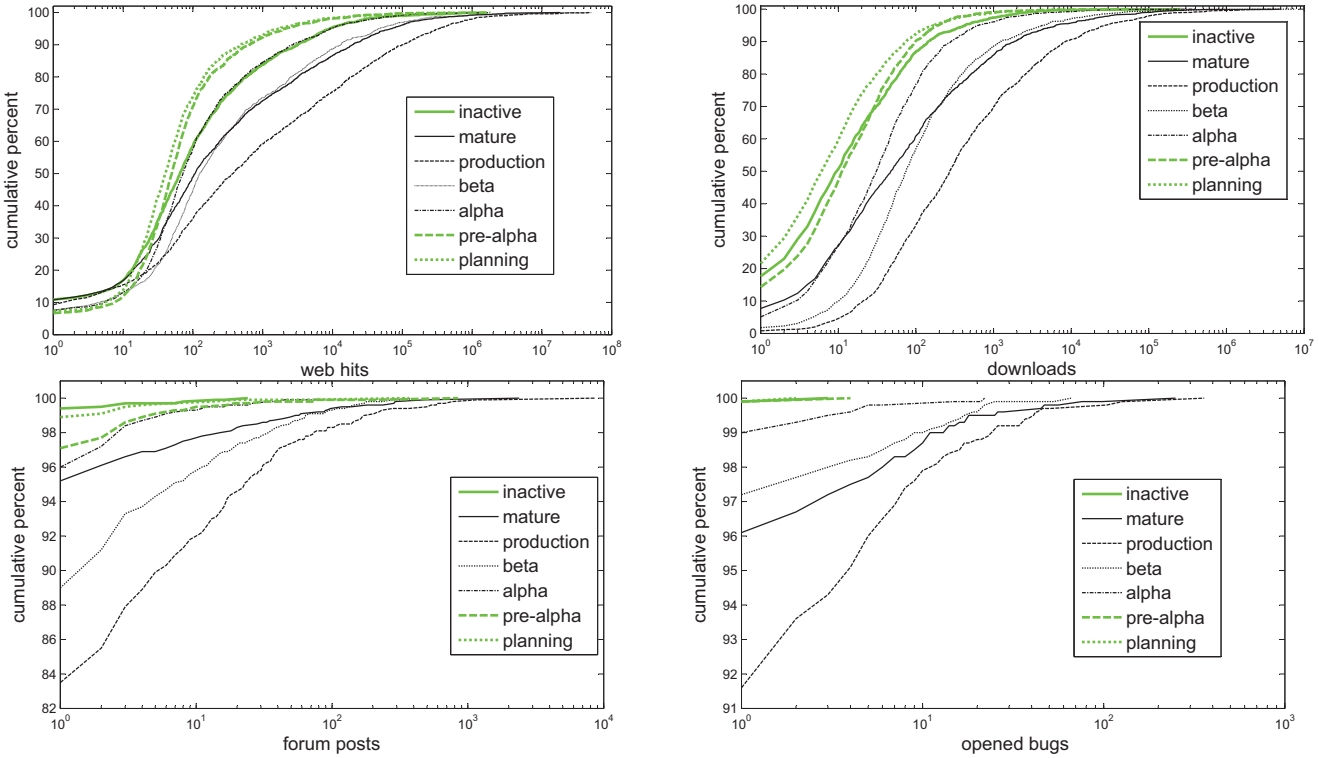


Fig. 2. Cumulative percents of user-related indicators

In existing studies, researchers typically select only successful projects as their subjects among the four categories of projects. Our definition is consistent with the definition of Crowston *et al.* [10]. When they define the success of OSS projects, they also take both users and developers into account as the two major audiences. We understand that developers and researchers can have different opinions on the success of OSS projects. For example, some projects are of high quality and are useful to a small group of users. Although our methodology does not put these projects into the successful category and researchers typically will not select these projects as research subjects, their users and developers can still believe that these projects are successful.

IV. Empirical Assessment

Based on our methodology, we conducted an assessment to address the research questions as follows:

- RQ1: To what extent are indicators that are extracted from successful OSS projects different from indicators that are extracted from other OSS projects?
- RQ2: To what extent do successful OSS projects represent the whole population of OSS projects?

Our first research aim leads to the first research question. To answer the research question, we present cumulative percents and Q-Q plots of all the indicators (RQ1). Our second research aim leads to the second research question. To answer the research question, we cluster collected OSS projects into four categories, and analyze clustering result (RQ2).

From SourceForge, our extraction tool extracted statistics from 30th July 2010 to 29th September 2010, and the extraction was completed within one week. For each development status, we randomly extracted statistic from 2,000 projects. During the extraction, we find that only few projects changed their development status during the week, and some projects do not have specific statistic. For example, some projects do not release any files, thus they do not have download numbers. We filter out the aforementioned two types of projects to eliminate the impacts from changed development status and missing statistics. The remaining 11,684 projects include 1,554 inactive projects, 1,672 mature projects, 1,556 production/stable projects, 1,631 beta projects, 1,672 alpha projects, 1,750 pre-alpha projects, and 1,849 planning projects. More details of our work can be found on our project site.[2]

TABLE II
Cumulative percents of web hits for inactive projects

Web hit	Frequency	Percent	Cumulative percent
0	161	10.4%	10.4%
1	8	0.5%	10.9%
...	...	...	...
1,327,487	1	0.1%	99.9%
1,523,322	1	0.1%	100.0%

A. RQ1: To what extent are indicators that are extracted from successful OSS projects different from indicators that are extracted from other OSS projects?

1) The cumulative percents of indicators: Figure 2 presents cumulative percents of user-related indicators. We build the figure based on distributions of these indicators, and we next use the inactive curve of web hits in this figure as an example to illustrate the building process. We organize sums of web hits by their frequencies. Table II shows distributions of web

[2]http://itechs.iscas.ac.cn/cn/membersHomepage/zhonghao/oss.html

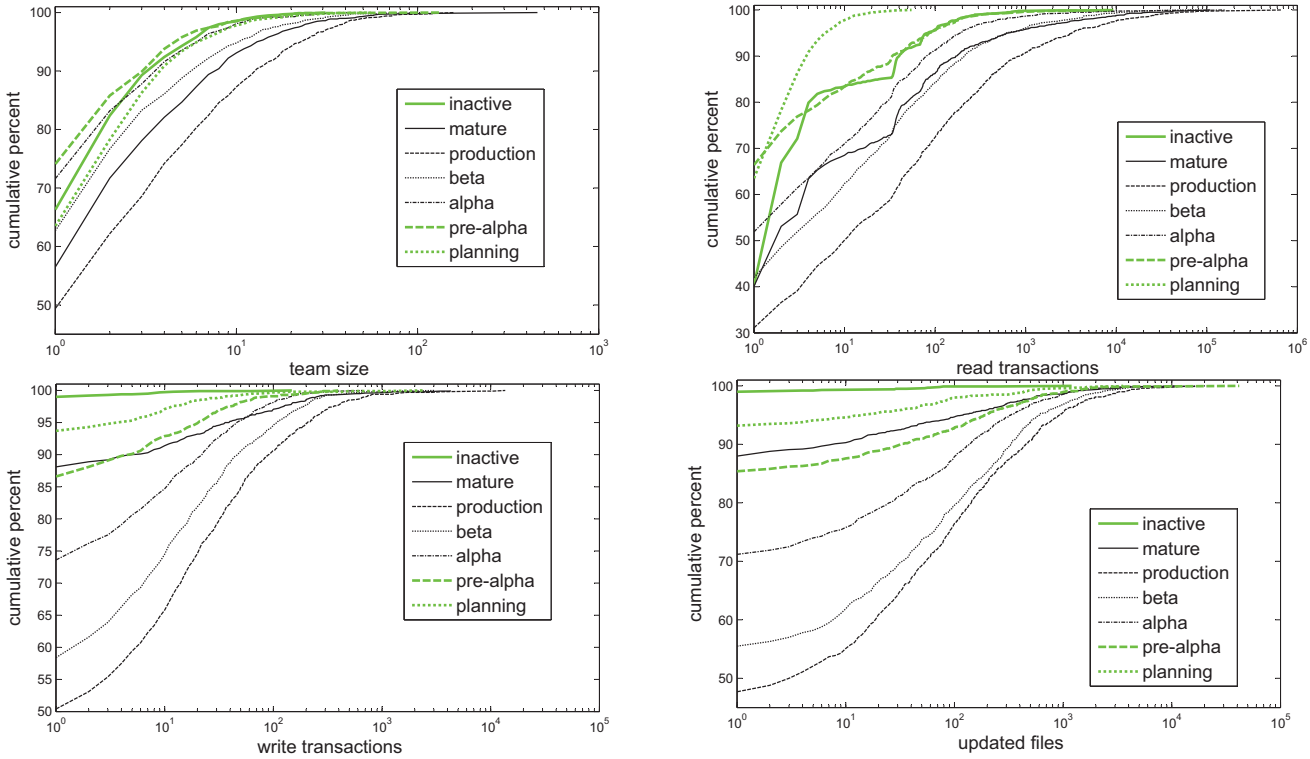


Fig. 3. Cumulative percents of developer-related indicators

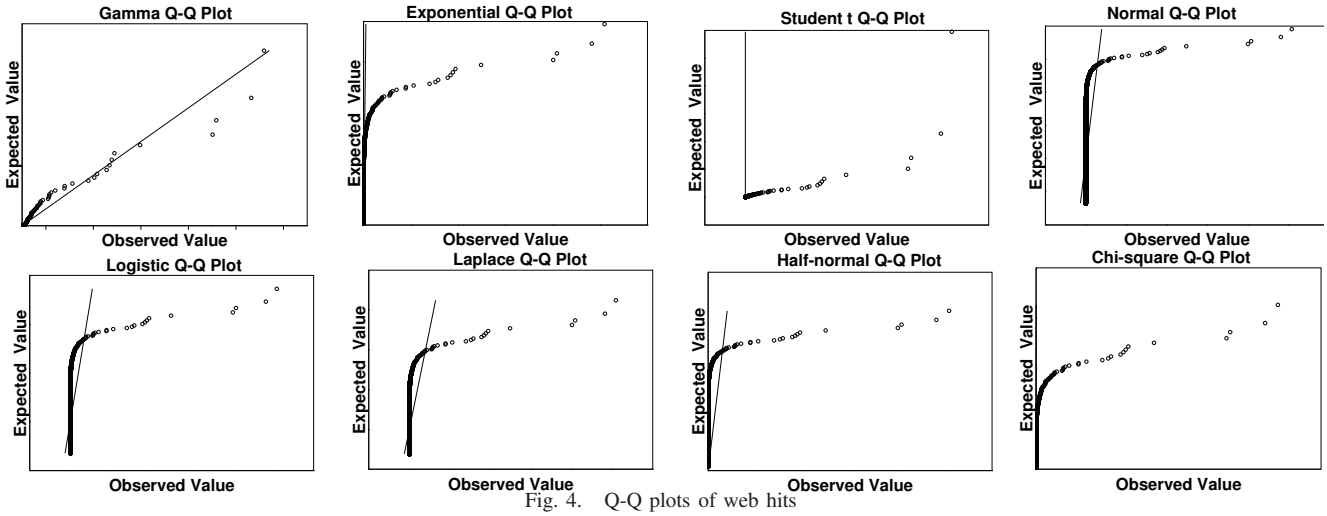


Fig. 4. Q-Q plots of web hits

hits for inactive projects partially. Column "Web hit" lists sums of web hits for inactive projects. Column "Frequency" lists frequencies of inactive projects with corresponding sums. Column "Percent" lists percents of corresponding frequencies. Column "Cumulative percent" lists cumulative percents of corresponding frequencies. For example, from the last row of Table II, only 1 inactive project has 1,523,322 web hits; the corresponding frequency is approximately 0.1% (1/1554); and the cumulative percent is 100% (*i.e.*, no inactive project has more than 1,523,322 web hits in the two months). To build the inactive curve as shown in Figure 2, we use Column "Web hit" in Table II to fill in the horizontal axis, and Column "Cumulative percent" in Table II to fill in the vertical axis. As several elite projects have much more web hits than the majority has, we choose the logarithmic scale instead of the linear scale for the horizontal axis of Figure 2 to better present web hits. Based on the result as shown in Figure 2, we come to the following findings:

Finding and implication 1: From Figure 2, we find that all the four indicators are highly unbalanced. Indicators of several elite projects are hundreds or even thousands larger than indicators of the majority. If researchers plan to draw conclusions from user-related indicators, they should be aware of this huge difference.

Finding and implication 2: We find that many users are willing to browse contents or download files for a trial run, but far fewer users will post on forums or open bug reports. The existing communication channels between users

and developers may be too complicated for users to participate, and developers should consider providing a more user friendly and attractive environment.

Finding and implication 3: We find that different development status has different attractions to users, and the attraction order to users is as follows: production, mature, beta, alpha, inactive, pre-alpha, and planning. However, development status can be of the different meanings as users would normally expect. For example, in contrary to expectations from many users, we find that some mature projects still have many bugs to be fixed.

Figure 3 shows cumulative percents of developer-related indicators. Comparing with the result as shown in Figure 2, we come to the following findings:

Finding and implication 4: We find that all the four developer-related indicators are also quite unbalanced. Several elite projects are hundreds or even thousand times more attractive to developers than the majority of projects. Our result is consistent with existing empirical studies. For example, Capiluppi *et al.* [7] reported that 49% projects have only one developer. Our result shows that about 50% projects have only one developer, and 90% projects have fewer than ten developers. Capiluppi *et al.* [7] consider that a team forms a community only when it has more than ten developers. Following their definition, we find that only 10% projects can have communities, since only these projects have more than ten developers. Thus, we conclude existing approaches on social networks of OSS projects (*e.g.*, the one proposed by Bird *et al.* [5]) are applicable for only 10% elite projects.

Finding and implication 5: We find that different development status also has different attractions to developers, and the attraction order to users is as follows: production, beta, alpha, mature/pre-alpha, planning, and inactive. Comparing with Finding 3, we find that users and developers can have different perceptions on specific development status, since the two orders are different.

2) The QQ plots of indicators.: In statistics analysis, Q-Q plots are commonly used to compare shapes of two distributions. In Q-Q plots, the horizontal axis and the vertical axis list quantiles of theoretical distributions and observed distributions, respectively. If two compared distributions are identical, the Q-Q plot follows 45-degree line. If two compared distributions agree after linearly transforming, the Q-Q plot follows some lines, but not necessarily 45 degrees.

To reveal which distributions all the indicators follow, we draw various Q-Q plots for the eight common distributions (*i.e.*, gamma, exponential, student t, normal, logistic, laplace, half-normal, and chi-square distributions). Figure 4 shows the drawn Q-Q plots of web hits. For the sub-graph of each distribution, the horizontal axis lists quantiles of the corresponding theoretical distribution, and the vertical axis lists quantiles of observed web hits. Due to space limit, we do not present Q-Q plots of other indicators. The Q-Q plots of these indicators are all similar with the Q-Q plots of web hits. From the Q-Q plots shown in Figure 4, we come to the sixth finding as follows:

Finding and implication 6: We find that all the indicators follow the gamma distribution, since only the Q-Q plots of the gamma distribution are linear. In particular, web hits and downloads are identical with the gamma distribution, and other indicators agree with the gamma distribution after linearly transforming. Here, the lower parts and the higher parts of some indicator may follow different distributions. We further discuss this issue in Section IV-D.

B. RQ2: To what extent do successful OSS projects represent the whole population of OSS?

Different indicators are not comparable. To combine all the indicators, we define the normalization function as follows:

$$norm(x_i) = \frac{\log(x_i + 1) - \log(\min(x) + 1)}{\log(\max(x) + 1) - \log(\min(x) + 1)} \quad (1)$$

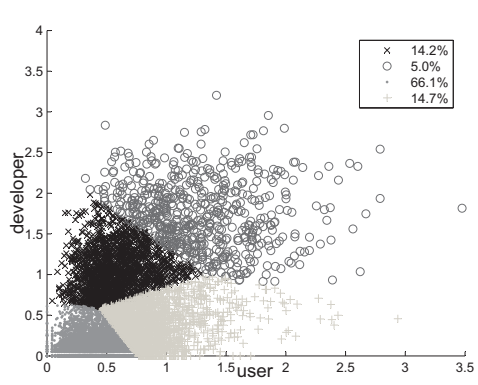
Based on the preceding two sums, we use the K-means clustering technique [37] to cluster all the projects. As we define four categories totally, we set the number of clusters as four. Figure 5 shows the result. In Figure 5, horizontal axes list attractions to users, and vertical axes list attractions to developers. For each figure, we use a legend to show percents of categories: "$\times$" denotes percents of developer-preference projects; "$\circ$" denotes percents of successful projects; "$+$" denotes percents of user-preference projects; and "$\cdot$" denotes percents of under developing projects. From the legend of Figure 5, we come to our seventh finding as follows:

Finding and implication 7: We find that 66.1% projects are under developing; 14.7% projects are user-preference; 14.2% projects are developer-preference; and only 5.0% projects are successful in total. Given the small percentage of successful projects, the sample size is not large enough to be representable to the whole population of OSS projects.

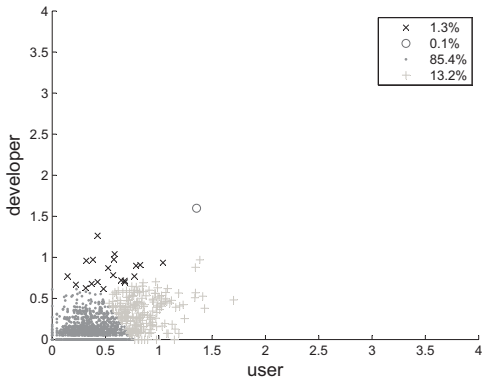
In Figure 5, we further calculate the clustering result of each development status, and we derive our eighth and ninth findings as follows:

Finding and implication 8: We find that users may have incorrect perception on some development status. For example, as shown in Figure 5, 63.2% mature projects are still under developing. Among them, some projects even have no developers, and are already abandoned by their developers. In some cases, developers may refer a project as mature, simply because they do not want to maintain the project any more.

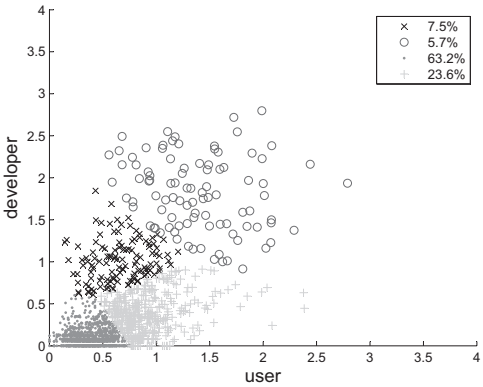
Finding and implication 9: We find that in general, a project is often not attractive to users or developers during its early development status such as planning, pre-alpha, and alpha. Gradually, some projects can become more attractive to developers and users in the follow-up development status, but the increase of user attractions are much slower than the increase of developer attractions. For example, 5.9% planning projects are developer-preference, and 6.5% planning projects are user-preference. At the same time, 11.6% pre-alpha projects are developer-preference, but only 6.9% are user-preference. The trend shows that users are slower to change than developers are. As another example, inactive projects also show similar trends, since 13.2% inactive projects are still user-preference even after they become inactive.



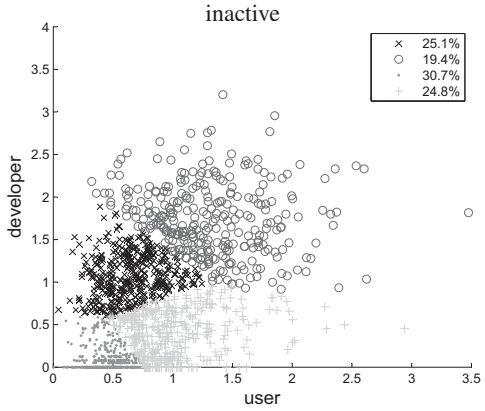
all



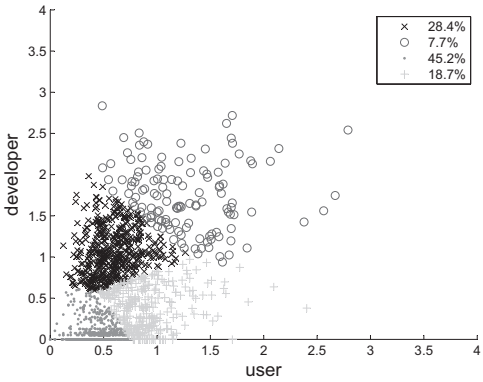
inactive



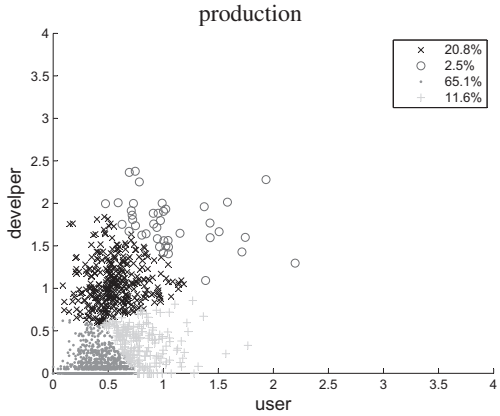
mature



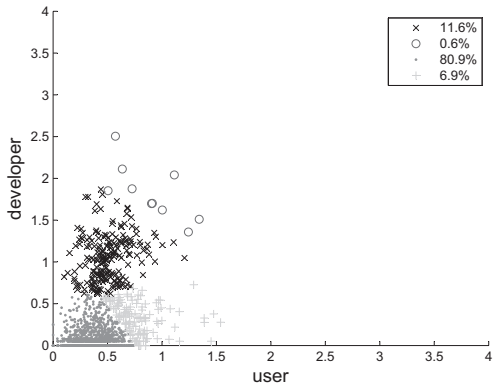
production



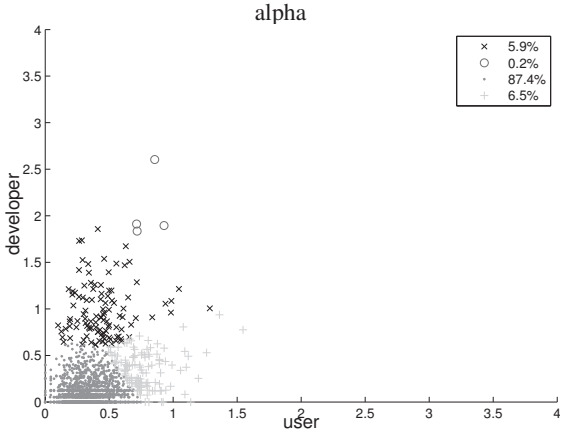
beta



alpha



prealpha



planning

Fig. 5. Clustering result

C. Summary

We summarize our findings as follows:

Result 1. Based on Finding 7, we conclude that successful OSS projects are not representative to the whole population of OSS, since successful projects are in the minority.

Result 2. Based on Findings 1, 4, and 6, we conclude that all the eight indicators follow the gamma distribution, and indicators of few successful OSS projects are much larger than the majority. The highly unbalanced distributions of analyzed indicators imply that conclusions drawn from successful projects can be quite different from conclusions drawn from the majority.

Result 3. Based on Findings 3, 5, 8, and 9, we conclude that different development status has different attractions to users and developers. In addition, we find that users and developers have different perceptions on development status in SourceForge. As the different perceptions mislead users, we suggest that these status should be clearly defined to eliminate differences.

Result 4. Based on Finding 2, we conclude that existing channels fail to attract users for participating OSS development. As user feedback is quite important for OSS development, we suggest that researchers and practitioners should provide better channels to collect user feedbacks.

D. Threats to validity

In this section, we discuss the briefly threats to validity that can affect our findings.

The threats to internal validity include that we choose limited indicators, and these indicators may not well reflect the success of OSS projects. The threats should be reduced if we introduce more indicators into analysis in future work. The threats to internal validity also include the setting of our study. For example, we set the duration for calculating sums as two months, and the impacts of other durations are not evaluated in our study. As another example, we analyze all the projects as a whole, and domains of projects may bias the analysis result. The threats should be reduced if we put projects of different domains into different categories, and analyze each category separately in our future work.

The threats to external validity include that we do not consider selected data and normalization techniques used in specific empirical studies. Our findings support that conclusions drawn from existing empirical studies may not be representative. However, a specific empirical study may use data other than our analyzed indicators, and may have techniques to reduce the huge differences of collected data. As a result, the impacts of our findings to existing empirical studies may vary and could not be fully determined. The threat should be reduced by revisiting existing empirical studies with data extracted from both successful projects and other projects. Like any other empirical studies, the threats to external validity include the time issue. Our result reflects the state of the art, and can become inapplicable years later with the rapid progress of the OSS development. The threat could be reduced by replicating the empirical study years later with update-to-date data and our proposed methodology.

V. Discussions and Future Work

Studies on unsuccessful projects. Our study reveals that most OSS projects are unsuccessful, but existing studies typically focus on only successful projects. With our findings, researchers can improve their work in two following directions. First, researchers can conduct studies on more unsuccessful projects to provide comprehensive understanding on the OSS phenomenon. Second, researchers can propose approaches that are effective for the predominate unsuccessful projects to achieve success. In future work, we plan to work towards both the two directions, so that we can provide more comprehensive empirical result and more effective approaches.

Distributions of indicators. Although all the indicators follow the gamma distribution as a whole, different ranges of a single indicator may follow different distributions, and we do not provide any theories to explain the phenomenon. We released all the data on our project website, so other researchers can also try different distributions, and propose their theories to explain why indicators follow such distributions.

Open source vs closed source. Our study provides no comparison between open source software and closed source software. Instead, our study provides insights on the subject selection of future studies to ensure the representativeness of their conclusions. We agree that a convincing comparison between open source software and closed source software may be achieved, if replicating our study with comparable numbers of closed source projects (e.g., with ISBSG data[3]). In future work, we plan to work towards this research direction.

Life cycles of OSS projects. Chillarege [9] presents the product life cycle of commercial software. In future work, we plan to investigate whether OSS projects follow similar life cycles, and to propose approaches that can predicate the development status changes of a given OSS project. In addition, our empirical study analyzes only one period of statistics from SourceForge. In future work, we plan to extract more statistics from different time period and compare these statistics to reveal the evolution of OSS projects in relation to their project success.

VI. Conclusion

There are many existing empirical studies that are conducted on several quite successful projects (*e.g.*, Linux, Mozilla, and Apache), and their conclusions heavily rely on representativeness of their selected subjects. As an attempt to investigate the true representativeness of these selected subjects, we conducted an empirical study based on an extensive collection of 11,684 projects hosted on SourceForge, allowing us to understand to what degree successful projects can be used to represent the entire OSS community.

An assessment methodology has been devised and employed in the study, the result shows only 5% projects are considered

[3]http://www.isbsg.org

truly successful, and indicators extracted from successful projects are quite different from indicators extracted from the majority. As a result, we can reasonably indicate that the conclusions produced by many of the empirical studies may be incorrectly representing the reality in the entire OSS community, techniques and methods derived from these studies may only be relevant to the selected successful projects, which is in fact the minority cases according to our assessment.

The statistics produced from the study are also important to reflect the current status of conclusions drawn from many OSS empirical studies, and can be used as a reference for researchers to determine the correct sample size and the limitations of selected project subjects.

ACKNOWLEDGMENTS

We appreciate reviewers for their supportive and constructive comments. This work is sponsored by the National Natural Science Foundation of China No. 61100071 and 60873072.

REFERENCES

[1] S. Ajila and D. Wu. Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software*, 80(9):1517–1529, 2007.

[2] L. Aversano, G. Canfora, L. Cerulo, C. Del Grosso, and M. Di Penta. An empirical study on the evolution of design patterns. In *Proc. ESEC/FSE*, pages 385–394, 2007.

[3] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: Bugs and bug-fix commits. In *Proc. 16th FSE*, pages 97–106, 2010.

[4] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced?: Bias in bug-fix datasets. In *Proc. ESEC/FSE*, pages 121–130, 2009.

[5] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proc. 16th FSE*, pages 24–35, 2008.

[6] A. Boulanger. Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal*, 44(2):239–248, 2010.

[7] A. Capiluppi, P. Lago, and M. Morisio. Characteristics of open source projects. In *Proc. 7th CSMR*, pages 317–327, 2003.

[8] E. Capra, C. Francalanci, and F. Merlo. An empirical study on the relationship between software design quality, development effort and governance in open source projects. *IEEE Transactions on Software Engineering*, 34(6):765–782, 2008.

[9] R. Chillarege. The marriage of business dynamics and software engineering. *IEEE Software*, 19(6):43–49, 2002.

[10] K. Crowston, H. Annabi, and J. Howison. Defining open source software project success. In *Proc 24th ICIS*, pages 327–340, 2003.

[11] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open source software development: What we know and what we do not know. *ACM Computing Surveys*, 44(2):1–35, 2012.

[12] B. Dagenais and M. P. Robillard. Creating and evolving developer documentation: Understanding the decisions of open source contributors. In *Proc. 18th FSE*, pages 127–136, 2010.

[13] T. Dinh-Trong and J. Bieman. The FreeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, pages 481–494, 2005.

[14] M. Gabel and Z. Su. A study of the uniqueness of source code. In *Proc. 16th FSE*, pages 147–156, 2010.

[15] D. M. German and M. D. Penta. Who are source code contributors and how do they change? In *Proc. 16th WCRE*, pages 11–20, 2009.

[16] M. Godfrey and Q. Tu. Evolution in open source software: A case study. In *Proc. ICSM*, pages 131–142, 2002.

[17] M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi. An empirical investigation into a large-scale Java open source code repository. In *Proc. 4th ESEM*, pages 1–10, 2010.

[18] V. Gurbani, A. Garvert, and J. Herbsleb. A case study of a corporate open source development model. In *Proc. 28th ICSE*, pages 472–481, 2006.

[19] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[20] M. Harman and P. McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering*, 36(2):226–247, 2010.

[21] A. Hindle, E. Barr, Z. Su, M. Gabel, and P. Devanbu. On the naturalness of software. In *Proc. 34th ICSE*, pages 837–847, 2012.

[22] C. Huntley. Organizational learning in open-source software projects: an analysis of debugging data. *IEEE Transactions on Engineering Management*, 50(4):485–493, 2004.

[23] C. Jensen and W. Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proc. 29th ICSE*, pages 364–374, 2007.

[24] M. Kim, D. Cai, and S. Kim. An empirical investigation into the role of refactoring during software evolution. In *Proc. 33st ICSE*, to appear, 2011.

[25] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. In *Proc. 10th ESEC/13th FSE*, pages 187–196, 2005.

[26] A. Ko, B. Myers, and D. Chau. A linguistic analysis of how people describe software problems. In *Proc. VL/HCC*, pages 127–134, 2006.

[27] S. Koch. Exploring the effects of sourceforge. net coordination and communication tools on the efficiency of open source projects using data envelopment analysis. *Empirical Software Engineering*, 14(4):397–417, 2009.

[28] A. Mockus, R. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[29] S. Olbrich, D. Cruzes, V. Basili, and N. Zazworka. The evolution and impact of code smells: A case study of two open source systems. In *Proc. 3rd ESEM*, pages 390–400, 2009.

[30] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language API descriptions. In *Proc. 34th ICSE*, pages 815–825, 2012.

[31] J. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4):246–256, 2004.

[32] N. Polikarpova, I. Ciupa, and B. Meyer. A comparative study of programmer-written and automatically inferred contracts. In *Proc. ISSTA*, pages 93–104, 2009.

[33] W. Scacchi. Free/open source software development. In *Proc. 6th ESEC/FSE*, pages 459–468, 2007.

[34] D. Schuler and A. Zeller. Assessing oracle quality with checked coverage. In *Proc. 4th ICST*, pages 90–99, 2011.

[35] L. Shi, H. Zhong, T. Xie, and M. Li. An empirical study on evolution of API documentation. In *Proc. FASE*, pages 416–431, 2011.

[36] S. Sowe, I. Stamelos, and L. Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems & Software*, 81(3):431–446, 2008.

[37] H. Spath. *The Cluster Dissection and Analysis Theory FORTRAN Programs Examples*. Prentice-Hall, 1985.

[38] S. Wang, D. Lo, Z. Xing, and L. Jiang. Concern localization using information retrieval: An empirical study on Linux kernel. In *Proc. 18th WCRE*, pages 92–96, 2011.

[39] M. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1, 1968.

[40] Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida. Collaboration with lean media: how open-source software succeeds. In *Proc. CSCW*, pages 329–338, 2000.

[41] Y. Ye and K. Kishida. Toward an understanding of the motivation Open Source Software developers. In *Proc. 25th ICSE*, pages 419–429, 2003.

[42] L. Zhang, S.-S. Hou, J.-J. Hu, T. Xie, and H. Mei. Is operator-based mutant selection superior to random mutant selection? In *Proc. 32nd ICSE*, pages 435–444, 2010.

[43] H. Zhong, L. Zhang, and H. Mei. An experimental study of four typical test suite reduction techniques. *Information and Software Technology*, 50(6):534–546, 2008.