Empirical Evidence on Developer’s Commit Activity for Open-Source Software Projects

Sihai Lin¹, Yutao Ma²*, Jianxun Chen¹

1. College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China
2. State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China

*e-mail: ytma@whu.edu.cn

Abstract—The manner of development is an important factor for the success of open-source software (OSS). Through mining the information of developer’s commits, researchers within the community of software engineering can investigate evolutionary aspects of OSS projects and analyze developer’s behaviors and collaboration. In this paper we conducted statistical analyses on commit activity for four OSS projects, and found that (1) the commit size in terms of new definitions roughly follows a power-law distribution, and exhibits self-similarity in the temporal dimension; (2) there are five common zones for the distribution of commit activity across various releases in terms of our indicator, and there exists an interesting “deadline effects” in the last zone (i.e. so-called rushing deadline); and (3) developers do prefer to fix bugs in the stage of rushing deadline, perhaps due to deadline pressure. These findings may provide a new insight into schedule planning, resource allocation and quality assurance of OSS projects.

Keywords—open-source software; commit; power law; release; self-similarity

I. INTRODUCTION

Over the last two decades, open-source software (OSS) has widely been used by individuals, companies, universities and governments all over the world. For example, as one of the prime examples of OSS, the Apache HTTP Server became the most popular web server software in use in April 1996, and as of December 2012 it was estimated to serve 63.7% of all active websites on the Internet [1]. Usually, the first perceived advantage of OSS is to lower software costs and simplify license management. Hence, more and more enterprises began to use OSS systems instead of commercial software to reduce IT budgets. A Zenoss survey published in 2010 revealed that 98% of the survey respondents indicated usage of open-source systems in their enterprises [2]. According to a report of the Standish Group, the adoption of OSS models has resulted in savings of about $60 billion per year to consumers [3].

OSS’s great potential to achieve success depends largely on the manner of development. An OSS system or project is typically built and maintained by a community of volunteer programmers or developers with a decentralized self-organizing organization structure. Since they are distributed among different geographic regions in the world and work for one or more OSS projects at different times, this demands for software tools that can assist their collaborative software development and facilitate source code management and conflict resolution. As we know, revision control systems such as Concurrent Versions System (CVS) and later Subversion (SVN) and Git are well-known examples of such software tools, which could help developers to centrally manage source code files and the changes to those files in the repository of an OSS project.

Apache SVN is such an OSS which can maintain current and historical versions of various types of files in a repository, and has been deemed as a compatible successor to CVS. Every OSS project (on the SourceForge, Google Code, etc.) has a history log for files’ each change and the related message ever committed to its SVN repository. Thus, through mining OSS developer’s commits, on one hand, researchers can investigate evolutionary aspects of an OSS system as well as its components; on the other hand, this could provide us a statistical way to analyze developer’s individual behaviors and teamwork, which might prove useful in quantifying and understanding the dynamics of human behavior on a collective scale [4]. Furthermore, such empirical evidence may offer new insights into schedule planning, resource allocation and quality assurance of an OSS system [5].

Although previous studies focused on commit size distribution [6,7,8], commit classification [6,9,10], developer’s contribution and expertise estimation [11,12,13], and task assignment [11,14], as far as we know, it is surprising that there are few representative statistical analyses on commit activity for OSS projects across a certain number of successive releases. To gain a deeper understanding of how an OSS project evolves, in this paper we will analyze 4 projects on the Apache.org in an attempt to answer the following questions: Are there any general rules within the duration between two adjacent releases (e.g. 2.1 and 2.2) that would be found in the whole lifecycle? And, does such an activity within the community of OSS developers have any characteristics similar to those found in professional software companies?

So, the goal of this paper is to explore what distinct feature commit activity possesses and how such an activity is distributed over different releases. Based on case studies, our research could provide empirical evidence for the evolutionary rules of OSS projects. Moreover, this would offer dedicated guidance for OSS developers to draw up project schedule and release a new version better. The main contributions of this paper are listed as follows.
(1) Investigating general statistical laws of commit activity for OSS projects in the whole lifecycle as well as within the duration between two adjacent releases;

(2) Identifying 5 common zones when analyzing the distribution of commit activity across various releases in the temporal dimension;

(3) Exploring the correlation between commit activity within the above-mentioned zones and software bugs fixing (and other types of development activities).

The remainder of this paper is structured as follows. Section II introduces related work. Section III explains the analysis method we followed, including research issues and data processing. In Section IV, we present the results of our sample of 4 OSS projects. Finally, Section V concludes this paper and puts forward future work.

II. RELATED WORK

For an OSS project, developers check out a working copy from its SVN repository, update files in the local workspace with the latest ones from the repository, and commit their changes to those local files to the repository. Commit is an important activity for OSS development, so there are a growing number of studies that offer some interesting findings on developer’s commits to OSS projects during software lifecycle.

Commit size distribution describes the probability that a given commit is of a particular size [8]. Hattori et al. found that the commit size in terms of the number of files follows a Pareto distribution [6], while Arafat et al. discovered that the commit size in terms of source lines of code (SLOC) follows a power-law distribution [7]. Such a distribution in terms of lines of code (LOC) is confirmed to be best described by a generalized Pareto model [8] similar to the finding in [7]. The commit size distribution with a long tail shows that developers might conduct large-size commits, though they are less likely to occur. Compared with the previous work, in this paper we will redefine commit size in terms of other indicators, and analyze their statistical distributions.

Because there is no recognized characterization of commit activity, its categorization is still vague. Hattori et al. proposed a classification framework in two dimensions (i.e. commit size and the comment of a commit) [6] to relate commits to certain types of activities such as code management and development. In [9], the research examined the version histories of 9 OSS systems to characterize a typical commit according to the number of files, the number of LOC, and the number of hunks committed together, and the results showed that the size categories of commits can be an indicator for the types of maintenance activities being performed. In order to build a categorization of commit types, Dragan et al. put forward an automated method by means of the meta-data for the stereotype information of methods added or deleted in a commit [10]. In this paper we plan to explore some characteristic rules recurred in various releases of an OSS project according to the proposed categorization [6].

Besides the above-mentioned studies, recent related research work focused on measuring developer contributions to an OSS project, and on recommending appropriate developers to tackle the changes to a specific file. Gousios et al. utilized the LOC that a developer commits to a SVN repository as a basic metric for development effort estimation [11]; Schuler et al. mined usage expertise from version archives to recommend experts for specific files [12], because developers who changed a file most often have the probable implementation expertise; considering that a developer who has actually contributed changes to specific files in the past will probably be a good choice of persons for their current or future changes, researchers proposed some new approaches [13,14] to assigning a task that performs software changes to a particular file to an appropriate developer or a ranked list of developers recommended. Actually, this paper has nothing to do with the studies in this field.

III. ANALYSIS METHODOLOGY

A. Research Issues

Like commercial software, OSS always tends to evolve through successive releases with incremental development. In a new release, the system will be updated by adding new functionality, removing redundant components or changing the existing ones, to deliver a set of required functionality or non-functionality such as performance, security and reliability. Changes to specific files committed by developers become an integral part of the system when a new release is delivered, and no other changes will be allowed after the delivery. Then, subsequent changes to files in the system’s SVN repository are going to be integrated into future releases. Because there are rarely previous studies taking release into account, in this paper we want to explore the following questions.

Q1: Are there any general rules of commit activity recurred both within the duration between two adjacent releases and in the whole lifecycle? In particular, does the distribution of commit size in terms of new indicators have some form of self-similarity in the temporal dimension?

Q2: Are there any common zones for the distribution of commit activity across various releases? That is to say, we want to detect whether there are any similar zones in which commit activities are active or inactive in terms of commit size.

Q3: If we do find such zones, are there any relationships between certain types of development activities and them? In other words, we are especially concerned whether these zones are related to bug fixing, code refactoring, and other activities.

B. Data Collection

Our methodology is based on case studies, so we selected four OSS projects written in Java on the Apache.org: Apache POI, Tomcat, Struts2, and Derby. The purpose of Apache POI is to create and maintain Java Application Programming Interfaces (APIs) for manipulating various file formats. Tomcat is an open source web server and servlet container developed by the Apache Software Foundation (ASF). Struts2 is an elegant, extensible framework for creating enterprise-ready Java web applications, and it uses and extends the Java Servlet API to encourage developers to adopt a model–view–controller (MVC) architecture. Derby is a rational database management
system (RDBMS) developed by the ASF that can be embedded in Java programs and used for online transaction processing.

**TABLE I. OSS PROJECTS ANALYZED**

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Date</th>
<th>Class</th>
<th>Commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>POI</td>
<td>APIs for manipulating file formats</td>
<td>2012-10-12</td>
<td>2,438</td>
<td>8,588</td>
</tr>
<tr>
<td>Tomcat</td>
<td>Servlet container</td>
<td>2012-10-12</td>
<td>1,980</td>
<td>14,481</td>
</tr>
<tr>
<td>Struts2</td>
<td>Framework for web applications</td>
<td>2012-10-12</td>
<td>1,521</td>
<td>9,999</td>
</tr>
<tr>
<td>Derby</td>
<td>RDBMS</td>
<td>2012-10-12</td>
<td>2,974</td>
<td>21,529</td>
</tr>
</tbody>
</table>

Table I shows the brief introduction to the projects analyzed, including the date we conducted our experiments, the number of class files and the total number of commits analyzed. These projects from different application domains were chosen to be experimental subjects based upon that they have been active for at least 2 years. For each project, we retrieved commit history of class files (from their respective main trucks of SVN repository) and release history for the whole system (from their respective websites) till the analysis date, and built sets of commit data within the duration between two adjacent releases according to release history. Moreover, we mined and collected bug information of these projects from Bugzilla and JIRA issues (http://www.atlassian.com/software/jira/overview).

### C. Data Processing

As for previous work about the distribution of commit size [6,7,8], the size was defined in terms of the number of files, LOC and other metrics. Here, we first present two new definitions of commit size as follow.

**Definition 1.** After a class file was created in a SVN repository, $RC$ is defined as the number of revisions to the class till its deletion. If a developer commits a change to a specific class from his/her local workspace, the value of $RC$ will increase by 1.

**Definition 2.** From the perspective of a project, $CN$ is defined as the total number of commits per time unit (e.g. one day, one week, or one month). It indicates the level of activity of a project during a given unit of time.

To answer the first question presented in the subsection III.A, we examined the distribution of commit size in terms of $RC$ and $CN$. Power function, exponential function, polynomial function and logarithmic function were utilized to fit release-level (i.e. the duration between two adjacent releases) and lifecycle-level data sets so as to indentify the best fitting curve and its corresponding function expression. Note that we used a cumulative distribution function (CDF) to reduce noise levels during the estimation of the scaling exponent of power function with the method introduced in [15]. It represents the frequency-of-occurrence of $m$ with value greater than or equal to a given number,

$$P_m(m) = \sum_{m' \geq m} P(m') \approx \int P(m') dm ' \sim m'^{-\alpha} \left( P(m) \sim m^{-\beta} \right). \tag{1}$$

In order to answer the second question, we identified common zones for the distribution of commit activity across various releases according to two parameters, namely the duration between two adjacent releases and $CN$. By normalizing both the former and the latter, their different values for the four OSS projects analyzed could be compared on a notionally common scale. The formulae of such normalization for these two parameters are described as follow.

$$tu' = f(tu) = \frac{tu}{D} = \frac{tu}{\sum tu}, \quad (2)$$

where $D$ is the duration between two adjacent releases, and $tu$ represents the $ith$ time unit within $D$.

$$cn' = g(cn) = \frac{cn - cn_{min}}{cn_{max} - cn_{min}}, \quad (3)$$

where $cn$ is the total number of commits during the $ith$ time unit, and $cn_{max}$ and $cn_{min}$ are the maximum and the minimum value of $cn$ within $D$, respectively.

However, for the projects analyzed, the distribution of $tu'$ is uneven for different pairs of adjacent releases. Hence, we used a simple partitioning method to cluster neighboring normalized time units across various releases. Then, the normalized total number of commits would be converted to another form of expression, which is the sum of $cn'$ within a new unified time unit for all releases of the 4 projects in question.

$$cn'' = h(cn') = \sum_{tu'} cn', \quad (4)$$

where $c$ is the unified time unit, and its value is equal to the unified normalized $D$ (whose value is 1 in this paper) divided by the number of equal parts $K$ (e.g. 100).

In addition, we analyzed the log message of commits within the identified common zones by using $Tf-idf$ (term frequency-inverse document frequency) to answer the third question. $Tf-idf$ is a numerical statistic which reflects how important a word is to a document in a collection or corpus [16].

$$w_j = tf_j \cdot \log_2 \frac{N}{n}, \quad (5)$$

where $w_j$ is the weight of term $T_j$ in document $Doc_n$, $tf_j$ denotes the frequency of term $T_j$ in document $Doc_n$, $N$ indicates the number of documents in corpus, and $n$ is the number of documents where term $T_j$ occurs at least once. The document $Doc_n$ is the set of all log messages of commits within $D$. Hence, we attempt to know what kind of activities developers could do within those common zones in terms of the terms in log messages of commits.

**IV. RESULTS AND DISCUSSION**

**A. Experimental results**

Besides the lifecycle of these four OSS projects, we analyzed 11, 10, 20 and 14 releases of POI, Tomcat, Struts2 and Derby, respectively. Due to space limitations, Table II shows the fitting statistics of a small number of samples, including lifecycle-level and two randomly-selected release-level cases. The power exponents for all release-level cases are
presented in Figure 1, where X axis denotes \( D \) and Y axis represents power exponent. The results show that all cases for \( RC \) roughly follow a power-law distribution, suggesting an obvious self-similarity. The observation indicates that most of classes are modified a few times, whereas the revisions to a small number of classes are very large within either the duration between two adjacent releases or the whole lifecycle. So, we will pay more attention to these frequently-modified classes in the evolutionary process of projects.

Then, we analyzed the distribution of commit size in terms of \( CN \) by using the same fitting method, where the time unit is set as day and week. L and R denote lifecycle and release, respectively; D and W represent time unit (i.e. one day and one week). Due to space limitations, Table III only shows the fitting statistics of a small number of samples. For each project, release-level cases are analyzed in terms of the same duration between any two adjacent releases used for \( RC \). The results show that all cases for \( CN \) roughly follow a power-law distribution, suggesting there is also an obvious self-similarity between lifecycle and releases. The observation indicates that the total number of commits in most of time units is small, whereas few time units contribute a large number of commits to its SVN repository.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Logarithmic</th>
<th>Polynomial</th>
<th>Exponential</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat (lifecycle)</td>
<td>( y = -353ln(x) + 1396.600 )</td>
<td>( R^2 = 0.712 )</td>
<td>( 0.161x - 25.095x + 820.030 )</td>
<td>( 0.478 )</td>
</tr>
<tr>
<td>POI (3.5betta4-beta5)</td>
<td>( y = 0.908ln(0.166x) )</td>
<td>( R^2 = 0.723 )</td>
<td></td>
<td>( 0.887 )</td>
</tr>
<tr>
<td>POI (3.5betta4-beta6)</td>
<td>( y = 0.134ln(0.153x) )</td>
<td>( R^2 = 0.657 )</td>
<td></td>
<td>( 0.913 )</td>
</tr>
<tr>
<td>POI (3.5betta4-beta5)</td>
<td>( y = 0.343ln(0.153x) )</td>
<td>( R^2 = 0.795 )</td>
<td></td>
<td>( 0.907 )</td>
</tr>
<tr>
<td>Struts2 (lifecycle)</td>
<td>( y = 0.201x + 0.230 )</td>
<td>( R^2 = 0.792 )</td>
<td></td>
<td>( 0.916 )</td>
</tr>
<tr>
<td>Struts2 (2.6-2.08)</td>
<td>( y = 0.744ln(0.127x) )</td>
<td>( R^2 = 0.856 )</td>
<td></td>
<td>( 0.917 )</td>
</tr>
<tr>
<td>Derby (lifecycle)</td>
<td>( y = 0.532ln(x) + 2036.2 )</td>
<td>( R^2 = 0.727 )</td>
<td></td>
<td>( 0.925 )</td>
</tr>
<tr>
<td>Derby (10.1.2-1.03)</td>
<td>( y = 0.532ln(x) + 2036.2 )</td>
<td>( R^2 = 0.727 )</td>
<td></td>
<td>( 0.925 )</td>
</tr>
<tr>
<td>Derby (10.1.2-1.03)</td>
<td>( y = 0.532ln(x) + 2036.2 )</td>
<td>( R^2 = 0.727 )</td>
<td></td>
<td>( 0.925 )</td>
</tr>
<tr>
<td>Derby (10.1.2-1.03)</td>
<td>( y = 0.532ln(x) + 2036.2 )</td>
<td>( R^2 = 0.727 )</td>
<td></td>
<td>( 0.925 )</td>
</tr>
</tbody>
</table>

**TABLE II. FITTING CURVES FOR RC**

**TABLE III. FITTING CURVES FOR CN**

According to the results of the distribution of commit size in terms of \( CN \), we further processed release-level data (whose time unit is one day) by normalizing and clustering to identify common zones among various releases. The unified \( D \) was evenly divided into 100 units, and \( CN \) was re-calculated based on the formula (4). The result for 55 releases of the 4 projects is illustrated by Figure 2, where X axis denotes time line, and Y axis represents normalized \( CN \). Taking into consideration the observation that \( CN \) follows a power-law distribution, we set a threshold to filter out trivial values of data points.
It is obvious from Figure 2 that commits within the zones I and III are less active than those of other zones. On the other hand, it is surprising that commits within the zone V are also active; furthermore, the average of commits within this zone is slightly larger than those of other active zones. This implies that with deadline approaching developers are still busy in preparing a new release, and the phenomenon what we found is similar to software engineers’ working overtime before the release of a new version to be delivered in professional companies. That is to say, although the date of a new release of an OSS project is not strictly fixed, within the zone V developers are still rushing out a new release which should be delivered in time, which is known as “deadline effects” [17].

Hence, according to the common zones identified in Figure 2, the duration between two adjacent releases (or so-called the period of a new release) could include 5 stages: preparation, active development, interim, active development, and rushing deadline. In the process of iterative and incremental development of software systems, the functional requirements for a new release should be validated and verified by software testing. In order to assure quality, OSS developers often locate and fix software bugs with the support of bug/issue tracking software such as Bugzilla and JIRA issues. Then, we want to explore the distribution of the activity of bug fixing over different stages, and to confirm whether developers tend to fix bugs in the stage of rushing deadline.

Because a box plot (also known as a box-and-whisker plot) can display differences between populations without making any assumptions of the underlying statistical distribution, we made use of it to analyze the data of commit log messages about bug fixing for the four projects in question. The result is displayed in Figure 3, where X axis represents four zones (i.e. II, III, IV and V) in Figure 2, and Y axis denotes the number of bugs fixed within the zone. The plot is interpreted as follows: the bottom and top of the box are the 25th and 75th percentile (the lower and upper quartiles, respectively), and the band near the middle of the box is the 50th percentile (the median); the “red star” is the outlier; and \( p \)-value attached to the plot expresses the probability that the observed difference in the number of bugs fixed among four zones is expected by chance.

Obviously, the difference in the number of bugs that have been fixed among zones is significant over the usual criterion of 99% confidence, which is calculated by using the Kruskal-Wallis test. Statistically significant differences illustrate an interesting finding for OSS project developers who prefer to fix bugs in the stage of rushing deadline, perhaps due to deadline pressure. But for active zones, the difference does not seem so significant. Besides bug fixing, what kinds of common activities developers would perform within these zones?

| TABLE IV. TERMS WITH HIGH TF-IDF WEIGHT WITHIN VARIOUS ZONES |
|-----------------|-----------------|-----------------|-----------------|
| POI | Tomcat | Struts2 | Derby |
| avoid | show | misname | compute |
| record | import | rename | contribute |
| clash | scan | revert | backup |
| typecast | trim | spend | alter |
| eliminate | replace | change | attach |
| mod | complete | build | return |
| apply | extend | redesign | boot |
| alleviate | sync | report | prepare |
| import | suggest | inject | grant |
| centralize | break | import | import |

Then, Tf-idf (see the formula (5)) was used to analyze the problem. Because it is related to the kind of development activities, we focus on verbs that have high weight in commit log messages. After some highly-frequent terms were filtered
out, a ranked list of terms could be obtained with a simple program. The result of top-10 terms with high weight is listed in Table IV. Considering that these projects analyzed are domain-specific and maintained by different developers, there are no verbs except “import” recurred in all projects, which demands for an elaborate investigation on the relationship between development activities and these common zones.

B. Threats to validation

Although we tried to diversify the characteristics of projects by carefully choosing four different OSS projects, some of our findings may not be generalized to other projects. In addition, these findings may not be suitable for industrial software systems, since OSS projects have particular characteristics different from those of commercial software.

In fact, an active OSS project evolves over time. The whole lifecycle presented in this paper is an approximate estimation of the real entire lifecycle. Even so, we argue that the power-law distribution of RC and CN for the period of development in the future still holds because of the well-known “Matthew effect”. The number of partition units K for the unified duration between two adjacent releases may influence the common zones that have been identified in this paper. So, we will seek to detect more accurate range and boundary of these common zones by analyzing more OSS projects. Another issue concerns the usage of log messages as the information of bug fixing, because there is lack of standardization for commit log messages. The method depending on the frequency of keywords to analyze the bug information in our study may be slightly biased.

V. CONCLUSION AND FUTURE WORK

Developers from all over the world can work together to develop an OSS project with the support of revision control systems such as CVS and SVN. Commit is an important activity of such OSS development, which attracted increasing attention from researchers within the community of software engineering. In this paper we conducted statistical analyses on commit activity for 4 OSS projects across a certain number of successive releases, and uncovered the following findings:

1. The commit size in terms of RC and CN roughly follows a power-law distribution, within both the duration between any two adjacent releases and the whole lifecycle, implying an obvious self-similarity in the temporal dimension;

2. We analyzed 55 releases of these 4 projects in question in terms of CN at the scale of one day, identified 5 common zones for the distribution of commit activity, namely preparation, active development, interim, active development and rushing deadline, and found an interesting “deadline effects” in the stage of rushing deadline, though the date of a new release of these projects is not strictly fixed;

3. We mined commit log messages and bug information from Bugzilla and JIRA, and found statistically significant differences in the number of bug fixed among 4 common zones, suggesting that developers do prefer to fix bugs in the stage of rushing deadline, perhaps due to deadline pressure.

The future work is to detect more accurate range and boundary of these common zones by analyzing more OSS projects, and to relate the characterization of these common zones to more types of development and maintenance activities to facilitate evaluating the spread of bugs.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China under grant Nos. 61272111 and 61100017.

REFERENCES