An Empirical Study on Inter-Commit Times in SVN

Qiuju Hou¹, Yutao Ma³,⁴,*, Jianxun Chen¹,², Youwei Xu⁵

1. College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China
2. Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan 430065, China
3. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China
4. WISET Automation Co., Ltd., Wuhan Iron and Steel Group Corporation, Wuhan 430080, China
5. Information Center, Wuhan Housing Security and Management Bureau, Wuhan 430015, China
*E-mail: ytma@whu.edu.cn

Abstract—Until now, centralized revision control systems such as Subversion (SVN) have been widely used in open-source software (OSS) development. Commit is a basic and important operation for revision control, and it has attracted the attention of a large number of researchers. As far as we know, few of prior studies investigated the distribution of inter-commit times (known as commit intervals), which reveals the development dynamics of an OSS project to some extent. To gain a better understanding on OSS development processes, we conducted an empirical study on two representative projects written in Java, and found that (1) the distributions of commit intervals in the two projects in question roughly follow power laws, with commit bursts (i.e., the revisions in a SVN repository are updated quickly over a period of time) and heavy tails, and (2) the working mode of SVN and (full-time or volunteer) developers’ work habits contribute to commit bursts, while active committer’s individual behavior (such as his/her tasks completion and illness) and long vacations are the primary factors that result in long inter-commit times. The findings could provide a new insight into schedule planning for OSS projects based on developers’ historical commit behavior.

Keywords-subversion; development dynamics; commit interval; burst; heavy tail

I. INTRODUCTION

Over the past two decades, open source as a new model has been deemed as a trend of effective software development to improve the function and quality of software [1]. In an open Internet-based environment, free and open source software (OSS) attracts the developers from all over the world to work together in a collaborative manner. For an OSS project with multiple developers, revision control (also known as version control) is an essential ingredient to software code management. Revision control systems such as CVS (Concurrent Versions System) and SVN (Subversion) are often centralized, with a single authoritative code repository, and check-outs and check-ins done with reference to such a central repository [2].

As we know, both centralized and decentralized revision control systems can track and provide control over changes to source code. Commit, also known as code contribution, is an important operation for these systems, since it tells a revision control system that a developer wants to make the change(s) final and available to all developers. In this sense, the development and maintenance process of an OSS project that is under centralized revision control could be regarded to be composed of a series of commits [3, 4]. The importance of mining historical commits of OSS projects is twofold: on one hand, it provides a new insight into the evolutionary aspects of an OSS project as well as its components [5, 6], which contributes to a better understanding of OSS development and maintenance process [7]; on the other hand, it offers a feasible and reliable way to investigate how the cooperation among collective developers promotes OSS development [8, 9, 10], which may facilitate the organization of project teams [11].

To the best of our knowledge, a majority of prior studies on commit focus on commit size distribution as well as commit classification. The former describes the probability that a given commit is of a particular size in terms of the number of files, LOC (Lines of Code), or other measures [3, 7, 8, 12], while the latter classifies commits according to their features and relates a given commit to certain types of software activities such as code management and bug fixing [8, 13, 14, 15]. Until now, very little attention has been paid to exploring the impact of developers’ collective commit behavior on OSS development and maintenance process using statistical methods [4, 10]. Actually, this is an important issue within the field of OSS research and practice [16].

In order to gain a deeper understanding of the interplay between developers’ collective behavior and OSS development process, the main goal of this paper is to conduct an empirical study on modeling the development dynamics of OSS projects hosted by a SVN server from the perspective of developer’s commit behavior. Moreover, we analyzed two representative projects written in Java on the Apache.org in an attempt to answer the following questions: 1) to what extent can the development dynamics of an OSS project be modeled in terms of developers’ commit behavior, and 2) if such a model does exist, what factors may affect its primary (statistical) features? We hope our empirical findings could offer a better understanding of OSS development and maintenance process, as well as novel ideas for the solution to the above-mention issues.

The remainder of this paper is structured as follows. Section II introduces related work. Section III addresses research questions, and explains the experimental methods we followed. Section IV presents primary results, and discusses the implications of our findings. Finally, Section V concludes this paper and puts forward future work.

II. RELATED WORK

A commit is a basic unit of work performed by a developer. Previous studies about commit size distribution found that the distributions of commit size in terms of specific measures roughly followed power laws [3, 7, 8, 12], implying that large-sized commits do exist, though they are less likely to occur. Meanwhile some of researchers began to categorize commits according to their features such as size and comment (also
known as log message) [8, 13, 14, 15], and found that the category of commit size was able to be a sound indicator for the types of maintenance activities being performed [14]. For example, Hindle et al. [15] found that large-sized commits were more corrective while small-sized commits were more corrective. However, none of these studies took the dynamics of developer’s commit behavior into account.

As mentioned before, the development of an OSS project could be deemed as a collaborative process of developers’ collective commit behavior [4, 10]. Human behavior, as one of the significant issues in science, has a history of about one century since the time of Watson [17]. Based on the increasing evidence from communication to entertainment and work patterns, Barabási et al. found that the timing of many human activities within these fields followed non-Poisson statistics, characterized by bursts of rapidly occurring events separated by long periods of inactivity [18]. Interestingly, such heavy-tailed distributions of inter-event times have also been demonstrated in computer science [19], e.g., email communication, website access, instant messaging, and Linux command logs. Recently, only a few of researchers began to investigate such a problem in terms of mining historical commits in OSS repositories [4, 5, 10, 11], and their work laid a good foundation for this paper.

III. RESEARCH QUESTIONS AND EXPERIMENTAL METHODS

It is worth noting that this paper uses the GQM (Goal, Question, Metric) method (http://en.wikipedia.org/wiki/GQM) to carry out an empirical investigation. In order to accomplish the research goal, research questions, the metrics associated with each question, and the experimental methods we used are described in detail in this section.

A. Research Questions

**RQ1:** to what extent can the development dynamics of an OSS project be modeled?

For an OSS project that is under centralized revision control, its (central) repository is indeed a special kind of file server, which can record the history of changes to every file. The development and maintenance process of the project is actually an overall picture of what has been happening in the repository [4]. Thus, the goal of the first research question is to investigate how we model the changes in the repository based on the metrics for developers’ commit behavior. Moreover, if such a model does exist, what are its primary (statistical) features? We argue this would provide a better understanding of general laws for the development of an OSS project.

**RQ2:** what factors may affect the primary features of the model, and what are implications for OSS development process?

Intuitively speaking, statistical features of the model obtained from RQ1 may be determined by those factors that affect developers’ commit behavior, e.g., work pattern, team organizational structure, as well as developer’s social properties such as nationality, occupation, hobbies and geographical location. For example, Singh found that the small-world developer collaboration networks positively affected the productivity of the member developers of an OSS project [20]. So, the goal of the second research question is to explore the factors influencing the model and their implications for OSS development process. We believe that this may facilitate a more reasonable schedule planning based on historical commits.

B. Metrics associated with the research questions

Each time the repository of an OSS project accepts a commit submitted by a SVN client, a new revision will be created to represent a “snapshot” of the entire repository tree instead of an individual file, with a unique natural number. The initial revision number is 0, and each subsequent commit successfully accepted by the repository increases the revision number by one [21]. Therefore, the history of revision logs sorted by the revision number in ascending order reveals the development process of the project in question in essence. To answer the above research questions, we make use of the following metrics for developer’s commit behavior.

**Definition 1.** For a project, commit interval is the time difference between two consecutive revisions in the repository that hosts the project [4].

**Definition 2.** For a project, commit frequency is the number of new revisions created in the repository in a given period of time [5, 10].

In general, commit frequency measures how often developers commit changes to the project they are in charge of, while commit interval describes how long a project will receive a new commit by a developer. The former is roughly in inverse proportion to the latter. That is to say, the smaller an average of commit intervals in a given period of time is, the greater the corresponding commit frequency becomes, indicating that the project under discussion is more active.

C. Experimental Methods

![Experimental Methods Diagram](image)

**Figure 1.** The process of our experiment

The overall process of our experiment is shown in Figure 1. Firstly, we extracted all historical revisions of a given project from its repository with a SVN client, and sorted them by the number in ascending order. Secondly, we calculated each commit interval of every pair of two consecutive revisions, and modeled the distribution of commit intervals by curve fitting methods. Thirdly, we sought the factors that may affect the primary (statistical) features of the model obtained in the above step, and presented their implications for OSS development.

In the second step, the distribution of commit intervals describes the probability that a given interval between two consecutive commits is of a particular length in terms of units of time. In probability theory and statistics, such discrete (probability) distributions can be represented in terms of probability mass functions or cumulative distribution functions (CDFs). Because the visual form of the CDF is, generally speaking, more robust than that of the probability distribution function against fluctuations due to finite sample sizes, especially in the tail of the distribution [22], in this paper we utilized the CDF and estimated the power-law exponent with the method in [22], so as to reduce noise levels. Besides power-law function, other common functions such as exponential function were also used to fit the discrete data in our data set.
In the third step, in order to determine the factors that may affect the distribution of commit intervals, we conducted an experiment to calculate the differences among each day of a week, as well as the difference between weekdays and weekend, with regard to commit interval and commit frequency. As we know, the median can be used as a measure of location when a distribution is skewed, when end-values are not known. Generally, in order to estimate the difference between two populations, a simple way is to compare their medians rather than means by using the standardized box plot. Moreover, in statistics, the Mann-Whitney U test [23] is a non-parametric test of the null hypothesis that two populations are the same. If a particular population tends to have larger values than the other, the null hypothesis is rejected, suggesting that their difference is statistically significant.

IV. PRIMARY RESULTS AND DISCUSSION

A. Data Collection

In this paper, our analysis is based on case studies, and we selected two OSS projects written in Java from the Apache.org, namely, Apache POI and Tomcat. Apache POI is used 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). These two projects were selected in that they are from different application domains, and each one is long-lived and active. Table I shows a brief introduction to the two representative projects, including the numbers of class files, commits and committers.

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Class</th>
<th>Commit</th>
<th>Committer</th>
</tr>
</thead>
<tbody>
<tr>
<td>POI</td>
<td>APIs for file processing</td>
<td>2655</td>
<td>2147</td>
<td>14</td>
</tr>
<tr>
<td>Tomcat</td>
<td>Servlet container</td>
<td>2595</td>
<td>9048</td>
<td>19</td>
</tr>
</tbody>
</table>

Note that, we mined just a few of the historical revisions from their SVN repositories between January 1, 2009 and September 20, 2013, and then calculated all intervals between every pair of two consecutive commits. For example, the line chart of commit intervals in Tomcat is presented in Figure 2, characterized by bursts of rapidly occurring commits separated by long periods of inactivity (more than 150 hours).

B. Result for RQ1: modeling the development dynamics of an OSS project in terms of commit interval

The log-log scatter plot of the distributions of commit intervals in Tomcat and POI is shown in Figure 3, where X axis represents the length of a commit interval in hours and Y axis indicates the probability that a given commit interval takes on a value less than or equal to x. Actually, the probability was calculated as the ratio of the number of commit intervals whose values are not greater than x to the total number of commit intervals under discussion. It is obvious from Figure 3 that the distributions of commit intervals in the two projects are heavy-tailed, implying that long inter-commit times do exist, though most of intervals are short, for example, about 80% of commit intervals in Tomcat are less than one hour.

Then, we made a careful analysis of (curve) fitting functions for the distributions of commit intervals (see Figure 3), shown in Table II (where $R^2$ is the goodness of fit). The fitting statistics indicate that the distributions in both of the two projects are best fitted by power functions, which suggests that they roughly follow power-law distributions. What we found reveals that a SVN repository often receives commits quickly over a period of time (called commit burst) except for a small number of ones for special reasons. The occurrence of commit bursts is mainly due to the working mode of centralized revision control [4] and developers’ work habits. That is, a central SVN repository stores only the latest version of each file, so that the changes to HEAD of the trunk committed by different developers are always completed in a short time to ensure that everyone is working on the up-to-date files. In the following sub-section, we will discuss the factors that contribute to long periods of inactivity.

<table>
<thead>
<tr>
<th>Projects</th>
<th>Fitting functions</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>POI</td>
<td>$y = 0.0009902x + 0.8018$</td>
<td>0.5367</td>
</tr>
<tr>
<td></td>
<td>$y = -7.7404x - 06x + 0.003088x + 0.7086$</td>
<td>0.8157</td>
</tr>
<tr>
<td></td>
<td>$y = 0.06774\ln(x) + 0.6064$</td>
<td>0.8469</td>
</tr>
<tr>
<td></td>
<td>$y = 0.7975e_0.3833x$</td>
<td>0.4992</td>
</tr>
<tr>
<td></td>
<td>$y = 0.5583x$</td>
<td>0.9254</td>
</tr>
<tr>
<td></td>
<td>$y = 0.001712x + 0.8732$</td>
<td>0.4886</td>
</tr>
<tr>
<td></td>
<td>$y = -5.202e -05x + 0.006887x + 0.7865$</td>
<td>0.7853</td>
</tr>
<tr>
<td>Tomcat</td>
<td>$y = 0.3833 \ln(x) + 0.806$</td>
<td>0.6731</td>
</tr>
<tr>
<td></td>
<td>$y = 0.8785e_0.1137x$</td>
<td>0.4716</td>
</tr>
<tr>
<td></td>
<td>$y = 0.7284x$</td>
<td>0.8568</td>
</tr>
</tbody>
</table>

Finding: The distributions of commit intervals in the two OSS projects in question can be best modeled by power laws, with commit bursts and heavy tails.

C. Result for RQ2: what may cause long periods of inactivity and their implications for software development

Although our prior work [4] conjectured that the delivery of a new release leads to a long inter-commit time, there is no evidence to support such a hypothesis. Intuitively speaking, this
is indeed one of the factors that cause long periods of inactivity. Besides the delivery of a new release, we guess weekend, holiday and active developer’s behavior also result in the occurrence of long commit intervals. Then, we are going to examine the influence of these factors on the length of commit intervals one after another, and then determine which factors are the primary ones.

Figure 4. Changes of average commit intervals over time

Firstly, the changes of average commit intervals per time unit over time are presented in Figure 4, where each vertical dashed line indicate the date that a new version of the project was released, extracted from the history of changes (also known as changes log) on the homepage of each project under discussion. Considering the different level of activity of the two projects, we selected month and week for POI and Tomcat respectively on purpose. It is obvious that there are several abnormal points that represent large average commit intervals in Figure 4. Thus, we carefully examined these anomalies. To our surprise, a small number of them (in POI) are caused by the delivery of a new release, but others are not.

As shown in Figure 5, for POI there does exist a period of inactivity that exceeds 100 hours after a new version was released. This implies that, on one hand, the delivery of a new release of an active project (such as Tomcat) has little effect on developers’ normal commits; on the other hand, it is not the primary factor that results in very long inter-commit times.

Secondly, we tested whether weekends lead to long commit intervals. Because the average interval of POI is over 48 hours (i.e., the length), we just conducted an experiment on Tomcat. The standardized box plots of commit frequency and commit interval are presented in Figure 6. According to the comparison among the median of commit frequency and commit interval in each day of a week, the difference on commit frequency between weekend and weekdays is rather obvious, while the median of commit intervals on Sunday is the smallest in a week. Note that (1) we ignored the days without a commit or with only one commit, and (2) we excluded any pair of consecutive commits that don’t happen in the same day.

Moreover, we confirmed the result of Figure 6 by using the Mann Whitney U test (under the significance level of 0.05). The symbol of M-X in Table III means two groups of data on Monday vs. the other day in a week (from Tuesday to Sunday). If we make the data of commit frequency or commit interval on Monday as a reference, for commit frequency, both M-Ss reject the null hypothesis; for commit interval, M-S (Sunday) rejects the null hypothesis. The finding indicates that their differences are statistically significant.

| TABLE III. MANN WHITNEY U TEST FOR MON. VS. THE OTHER DAY |
|-----------------|------|------|------|------|------|------|
| Commit frequency | M-T  | M-W  | M-T  | M-F  | M-S  | M-S  |
| P               | 0.1440.0960.0810.3950.001<0.001|
| Z               | 1.4611.6651.8540.851-3.410-4.593|
| Commit interval  | M-T  | M-W  | M-T  | M-F  | M-S  | M-S  |
| P               | 0.7990.1590.1110.3610.1350.001|
| Z               | 0.2541.4101.5930.9141.495-3.323|
Because two consecutive commits sometimes do not occur in the same day, it may affect the result of our experiment. Hence, we re-classified the data into two groups, namely *Weekdays* and *Weekend* to reduce such a bias. That is, all pairs of two consecutive commits that occur on weekdays (from Monday to Friday) were categorized into one group. The similar results are presented in Figure 7 and Table IV, indicating that the commit frequency and commit interval over a weekend are different from those on weekdays.

![Figure 7. Standardized box plots for weekdays and a weekend](image)

### TABLE IV. MANN WHITNEY U TEST FOR A WEEKEND VS. WEEKDAYS

<table>
<thead>
<tr>
<th></th>
<th>P-value</th>
<th>Z-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit frequency</td>
<td>&lt; 0.001</td>
<td>-9.068</td>
</tr>
<tr>
<td>Commit interval</td>
<td>0.004</td>
<td>-2.876</td>
</tr>
</tbody>
</table>

The statistics for the SVN repository of Tomcat show that there are only 19.4% of commits completed on weekends, which accounts for low commit frequency values shown in the above figures. To our surprise, we also found that the group *Weekend* tends to have smaller values of commit interval than the other group *Weekdays*, perhaps due to that developers feel inclined to finish the task of committing a small number of changes at a certain time, and then have a rest in the remainder of the weekend. This may cause periods of inactivity. So, we selected the top 5% of long commit intervals in Tomcat (see the heavy tail in Figure 3), and found that 31.4% and 27.3% of these intervals occur on weekdays and weekends respectively. Interestingly, the ratios of the number of long commit intervals on weekends and weekdays to the corresponding total number of intervals are 5.24% and 1.19%, respectively. This finding indicates that for active OSS projects weekends do affect the normal development process and would result in large commit intervals (less than 48 hours) as a result of weekend breaks.

Thirdly, since the developers of the two projects are from the West, we selected three relatively long holidays, namely Christmas, New Year’s Day and Thanksgiving Day, to prove our conjecture. As New Year’s Day is too soon after Christmas, they could be deemed as one vacation lasting from December 24 to January 6. For POI and Tomcat, the longest commit intervals in hours during the two long vacations (i.e., Christmas & New Year’s Day and Thanksgiving Day) are listed in Table V, where *CN* and *T* represent Christmas & New Year’s Day and Thanksgiving Day, respectively. As we expected, these intervals fall into the top 5% of long commit intervals, implying that long vacations do affect the regular development and maintenance processes of OSS projects and result in long periods of inactivity.

![Figure 8. Comparison of commits per month between different committers](image)

Finally, we analyzed the influence of active developer’s behavior on commit intervals. Our prior work [4] found that a minority of committers contribute to the vast majority of commits. Hence, active committers can be defined in terms of the number of commits. In this paper, active committers were selected as the top three committers who contribute the most number of commits. Figure 8 presents a comparison of the number of commits per month between active committers and all committers. The curve about active committers is roughly in coincidence with that of all committers, suggesting that active committers play an important role in the development and maintenance of the two projects.

As we know, a commit interval between two consecutive commits is related to two committers. Then, according to the type of committers, we divided the top 5% of long commit intervals into three types of intervals: (1) a commit interval that two committers (active or not) are the same person is classified into group A, (2) if two committers are active and the same person, a commit interval is classified into group B, and (3) a commit interval belongs to group C in case two committers are both active regardless of whether they are the same person.

Table VI shows the proportions of the number of a specific category of commit intervals to the total number of the top 5% of long commit intervals. To our surprise, the proportion of the type C is considerable, especially for Tomcat. This indicates that active committers’ collective behavior can influence the length of commit intervals, though they have different tasks, backgrounds, habits and expertise. Interestingly, the top three longest intervals in both POI and Tomcat are all included in the group C. Very long inter-commit times (e.g., several months) between two consecutive commits to the projects in question do exist, perhaps because developers lose interest in the
projects, go on vacation, complete their tasks and wait for new tasks, and other accidental events.

<table>
<thead>
<tr>
<th>Projects</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>POI</td>
<td>46.73%</td>
<td>39.25%</td>
<td>71.96%</td>
</tr>
<tr>
<td>Tomcat</td>
<td>49.16%</td>
<td>46.37%</td>
<td>87.59%</td>
</tr>
</tbody>
</table>

**Finding:** The working mode of SVN and developers’ work habits contribute to commit bursts, while active committer’s individual behavior and long vacations are the primary factors that result in long inter-commit times.

### D. Threats to Validity

Because the results of this paper were obtained based on two case studies of OSS projects written in Java, they might not be generalizable for closed-source projects. Moreover, it is possible that we accidentally selected the two projects that exhibit such characteristics and regular patterns. Thus, we have to validate the generality of our findings based on more OSS projects. On the other hand, whether the results are still suitable for decentralized control system such as Git or not need to be further proved.

### V. Conclusion

This paper conducted an empirical study on two OSS projects on the apache.org that are under centralized revision control in terms of commit interval, i.e., the time difference between two consecutive commits. In summary, our primary findings and contributions are described as follows:

1. The distributions of commit intervals in both POI and Tomcat roughly follow power laws, with commit bursts and heavy tails, implying very long inter-commit times do exist, though a vast majority of commits are completed in a short time. This finding validates Barabási’s conclusion [18] within the field of software development, and indicates that the development dynamics of an OSS project can be modeled by means of developers’ commit behavior;

2. We carefully examined the top 5% of long commit intervals using statistical methods, and found that active committer’s individual behavior (unpredictable factor) and long vacations (regular factor) are the primary factors that cause very long periods of inactivity. This finding implies that the administrator(s) of an OSS project should pay more attention to active committers’ accidental events and regular holidays, and make a reasonable schedule after their departure.

Because many OSS projects have become fully or partially commercial [11], the future work is to design an algorithm for mining anomalies [24] in commit time series, so as to optimize software processes and improve development efficiency.

### ACKNOWLEDGMENT

This work is supported by the National Basic Research Program of China (No. 2014CB34040), the National Natural Science Foundation of China (Nos. 61272111 and 61273216), the Youth Chenguang Project of Science and Technology of Wuhan City in China (No. 2014070404010232), and the open foundation of Hubei Provincial Key Laboratory of Intelligent Information Processing and Real-time Industrial System (No. znss2013B017).

### REFERENCES