Distribution and Continuity of Developers' Contributions in OSS Projects: A Case Study

Zhongjie Wang[1], Dewayne E. Perry[2], Xiaofei Xu[1]

1 Harbin Institute of Technology, Harbin, China 150001
2 The University of Texas at Austin, Austin, TX, USA 78712
rainy@hit.edu.cn, perry@ece.utexas.edu, xiaofei@hit.edu.cn

Abstract– **Open Source Software (OSS) is usually developed by geographically-distributed developers in a collaborative manner. Different developers exhibit different behaviors and make diversified contributions to OSS projects. Objective of this paper is to discover individualized characteristics and common patterns of how developers contribute to OSS projects. Continuity is used to delineate how a developer actively contributes to the project over time. Case studies on two OSS projects reveal some significant phenomena on the distribution of developers' contributions relative to absolute time and relative to the milestones (i.e., releases) of OSS projects. We have found that OSS developers' contributions exhibit the "temporal locality", and most of the releases of an OSS project are dominated by the contributions of a limited number of developers.**

Keywords– Open Source Software; social development; developers' contributions; continuity; temporal distribution

1. Introduction

Different developers have different social and technical backgrounds. They joined an OSS project with different motivations [1, 2]. Consequently, the degree of their participation and the level of their contributions are diversified [3]. A set of individualized characteristics on the behaviors of OSS developers that distinguish one from others who belong to the same OSS project team do indeed exist [4].

Most of the recognized characteristics are closely related to externally visible behaviors that developers exhibit in the history of OSS projects. This paper is focused on exploring the personalized contributing styles of developers from a temporal dimension. A developer initiatively enforces to make contributions to OSS projects, especially code commits. Targeting at the contributions, a metric called "continuity" which delineates how frequently and consecutively a developer makes contributions to a project and is measured by the times, contributed Line of Code (LoC), and temporal distribution of his active behaviors, is studied. We introduce an approach of identifying and measuring the continuity of contributions from public data in OSS repositories, in which the continuity is refined into a set of fine-grained metrics based on the temporal distribution of the active behaviors of each developer, with respect to the absolute timeline (defined by calendar time) and the relative timeline (defined by a series of releases, or milestones, of an OSS project). Entropy is employed to give an overall measurement of the continuity. Case studies are conducted on two OSS projects (`JUnit` and `Guava`) hosted on GitHub. The identified characteristics are visualized, and developers are compared in terms of these characteristics to discover the difference and similarities among their behaviors.

This study tries to answer three research questions:

RQ1: How are a developer's contributions distributed on the absolute timeline?

RQ2: How are a developer's contributions distributed relative to the milestones of the OSS project?

RQ3: Are there distinct characteristics of different developers on the distribution/continuity of their contributions?

By case studies we find that, (1) Contributions of a developer often show temporal locality, and there are some long intervals that split a developer's contributions into stages; the more contributions a developer makes, the more evenly his contributions are distributed over time, with averagely shorter intervals; (2) Most of the releases of an OSS project are dominated by contributions of a limited number of developers, respectively, i.e., the active behaviors of a few developers contribute most of the changes of a release relative to its previous one. These would help OSS project coordinators get a good understanding on the working habits of developers in the project team, thus facilitate proactive and accurate project management activities such as task allocations.

The remainder is organized as follows. Candidate projects and the independent/dependent variables of the case study are presented in Section 2. Section 3 is the study approaches and results, followed by threats of validity in Section 4. Section 5 and 6 are the related work and conclusions.

DOI reference number: 10.18293/SEKE2016-008

2. Study Setup

2.1. Candidate OSS Projects

Two OSS projects from GitHub, `JUnit`[1] and `Guava`[2], are selected for the case study. They are diversified in many perspectives. From the perspective of time to live, `JUnit` is much older for above 14 years and `Guava` is for nearly 6 years. From the perspective of scale, `Guava` is larger with more than 1,700 source files and above 400K LoC, and `JUnit` is a medium project with 300-400 files and 40K LoC. From the perspective of team size, `JUnit` has a larger team with more than 130 developers, while `Guava` has a relatively smaller team. From the perspective of behavior intensity, `Guava` has the larger number of commits than `JUnit`, and on average, although `Guava` has a smaller team, its developers seem more active and commit more average LoC than `JUnit`.

2.2. Variable Selection

2.2.1 Independent Variables (IVs)

An OSS project has a team T and a sequence of code commits CS. Every time a developer submits code to the project repository, does he carry out an active behavior explicitly exhibited as the expansion or modification of existing source code in two forms: addition and deletion.

A commit $c \in CS$ is defined as $c = \langle PC, F, d, t\rangle$, where PC is c's parent commits on which c's code changes are based, and each commit might have 0, 1 or multiple parent commits; F is the files contained in c, and $\forall f \in F$, $l^+(f)$,$l^-(f)$ are the code lines added into and deleted from f, respectively, compared with the corresponding files in PC; d is the developer who authored the code changes in c; and t is the commit time of c.

Commits of a project form a sequence in which they are sorted by commit time in chronological order. In terms of a developer d_i, all the commits authored by d_i form a sub-sequence of CS and is denoted by CS_i. Multiple developers alternately check out codes from repository and commit changes back, so their commit sequences are interwoven with each other. Denote $\varphi_i = |CS_i|$, and $\sum_{i=1}^{\tau} \varphi_i = |CS|$.

In GitHub, a set of commits could be grouped together as a release being a phase achievement of the project[3]. Here we simply define a release r_j as a set of commits (the number of commits is $|r_j|$). Suppose there are totally γ releases in a project and they form a sequence $\langle r_1, ..., r_\gamma\rangle$ in terms of the release date.

A set of `git` commands are used to collect above-mentioned variables from `git` repositories, such as `git rev-list`, `git diff`, `git show`, `git for-each-ref ''refs/tags''`, etc.

2.2.2 Dependent Variables (DVs)

The dependent variables (DVs) are constructed in terms of two timelines: (1) The absolute timeline defined by calendar time; (2) The relative timeline defined by the date of releases;

For the former, the lifespan of a project is split into N number of timeslots $\langle p_1, ..., p_N\rangle$. The length of a timeslot may be 1 day, 1 week, 1 month, or any length of time. The following are the DVs used under the absolute timeline:

(1) $V_{times}(d_i)$,$V_{loc}(d_i)$: a developer d_i's distribution vectors of the times and LoC of active contributing behaviors in N timeslots, i.e., $V_{times}(d_i) = \langle \varphi_{i1}, \varphi_{i2}, ..., \varphi_{iN}\rangle$, $V_{loc}(d_i) = \langle l^+_{i1}, l^+_{i2}, ..., l^+_{iN}\rangle$.

We define $CS_{ik} = \{c_j | c_j \in CS_i \wedge t(c_j) \in p_k\}$ as the commits that belong to CS_i (the commit sequence of d_i) and fall into the timeslot $p_k (1 \leq k \leq N)$. Then, $\varphi_{ik} = |CS_{ik}|$ is d_i's times of active behaviors occurring during p_k; $l^+_{ik} = \sum_{c_j \in CS_{ik}} l^+(c_j)$ is the total LoC that d_i contributes during p_k, in which $l^+(c_j) = \sum_{f \in F(c_j)} l^+(f)$ is the sum of "additions" contained in the files of c_j, indicating the d_i's total contribution made in the commit c_j.

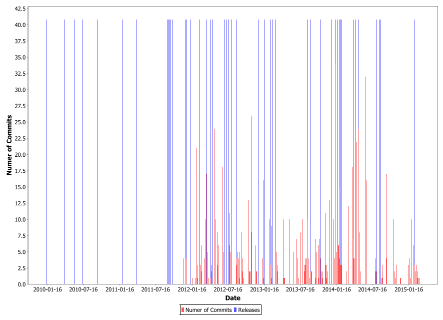
(2) $V_{interval}(d_i)$: the vector of intervals between two consecutive active behaviors of d_i, i.e., $V_{interval}(d_i) = \langle \delta_{i1}, ..., \delta_{i(\varphi_i - 1)}\rangle$ where $\delta_{ij} = t(c_{i(j+1)}) - t(c_{ij})$ measures the time interval between c_{ij} and $c_{i(j+1)}$ both included in d_i's commit sequence CS_i.

(3) $H_{times}(d_i)$,$H_{loc}(d_i)$: the continuity entropy of active contributing behaviors of a developer d_i. Entropy has been widely applied in software engineering for various measurement on software entities and software development activities, such as software complexity entropy [5] and software change entropy [6]. Here we borrow this concept to measure how a developer's contributions on source code are distributed over time. The first entropy is "times entropy" measuring the uncertainty in the times distribution of d_i's active behaviors in N timeslots $\langle p_1, ..., p_N\rangle$, i.e., $H_{times}(d_i) = -\sum_{k=1}^{N} (\frac{\varphi_{ik}}{\varphi_i}) \log(\frac{\varphi_{ik}}{\varphi_i})$. The second one is "LoC entropy" which measures the continuity of the LoC in d_i's active behaviors, i.e., $H_{loc}(d_i) = -\sum_{k=1}^{N} (\frac{l^+_{ik}}{l^+_i}) \log(\frac{l^+_{ik}}{l^+_i})$ where $l^+_i = \sum_{c_j \in CS_i} l^+(c_j)$ is the total LoC that d_i has contributed during the lifespan of the project.

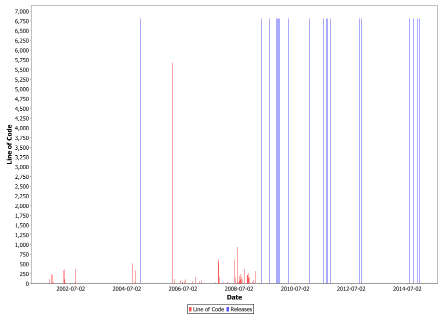
For relative timeline, $V(r_k)$ delineates the relative distribution characteristics w.r.t. the releases of a project:

(4) $V(r_k)$: the vector of the contribution ratios of all the developers in one release period r_k, i.e., $V(r_k) = \langle \sigma_{1k}, \sigma_{2k}, ..., \sigma_{\tau k}\rangle$ where σ_{ik} is the ratio of d_i's contribution (LoC) relative to the total contributions that all developers have made during the period from the release r_{k-1} to r_k, i.e., $\sigma_{ik} = \frac{l^+_i(r_k)}{l^+(r_k)}$, and $l^+_i(r_k) = \sum_{c_j \in r_k \wedge c_j \in CS_i} l^+(c_j)$, $l^+(r_k) = \sum_{c_j \in r_k} l^+(c_j)$,$\sum_{k=1}^{\tau} \sigma_{ik} = 1$, and $\tau = |T|$ is

[1]https://github.com/junit-team/junit
[2]https://github.com/google/guava
[3]https://help.github.com/articles/creating-releases/

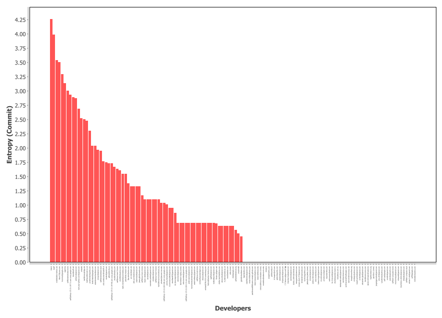


(a) $V_{times}(d_1)$ of Guava.

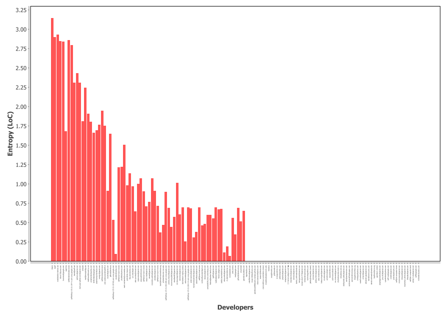


(b) $V_{loc}(d_2)$ of JUnit.

Figure 1. Barcode visualizing contribution distribution w.r.t. absolute timeline



(a) H_{times} of JUnit.



(b) H_{loc} of JUnit.

Figure 2. Comparison of the continuity entropy among developers

the total number of developers.

3. Study Results

3.1. Contribution Distribution w.r.t. Absolute Time

To exhibit the distribution of developers' contribution w.r.t. the absolute timeline, "barcode" is employed to visualize the distribution vectors $V_{times}(d_i)$ and $V_{loc}(d_i)$ for each developer d_i.

Fig. 1 gives the barcode of two developers from `JUnit` and `Guava`, respectively. Fig. 1(a) is the times distributions of a `Guava` developer, and Fig. 1(b) is the LoC distribution of a `JUnit` developer. The timeline is split into N timeslots (the length of timeslot is 1 day; N=5,235 for `JUnit` and N=2,003 for `Guava`). The red vertical bars are active contributions, and the height of the bars shows the times or LoC of behaviors occurring in the specific timeslot. As a reference, blue bars represent the releases of the project.

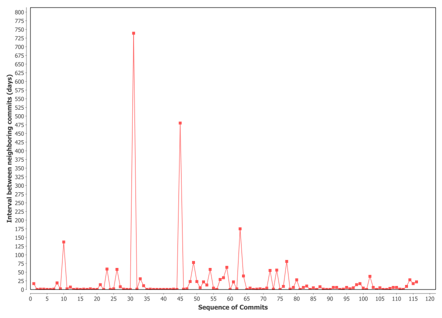
Barcode is a straightforward way to visualize the temporal distribution of developers behaviors. Without more precise quantitative metrics, the continuity and the intensity of active behaviors at different times can be intuitively observed from the barcode. For example, d_1 has intensive and continuous active behaviors from the middle of the project, d_2's active behaviors are split by two long intervals, and d_3's behaviors are concentrated in a very short period.

By observation of the barcode for 133 developers in `JUnit` and 44 in `Guava`, we find that the shapes of barcode of different developers are quite diversified. Result of Chi-squared test of independence has shown that there are significant difference between the distribution vectors of any pair of developers belonging to the same project. It confirms that the distribution vectors of active behaviors can be used as individualized characteristics of OSS developers.

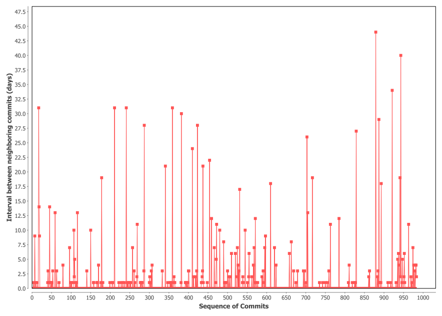
Fig. 2 demonstrates the continuity entropy H_{times} and H_{loc} of developers in `JUnit`. Developers are sorted by H_{times} in descending order. A higher entropy indicates that the developer's contributions are distributed over time in a more balanced way. The entropy of some developers in the right equals to 0 because all their active behaviors concentrate in only one timeslot.

From the comparisons between H_{times} and H_{loc} for `JUnit`, and between the ones for `Guava`, the distributions of H_{times} and H_{loc} show almost the same shape. The Spearman's rank-order correlation test shows that the two entropy are highly correlated. For `JUnit`, the correlation coefficient is 0.863 (p <0.01); for `Guava`, it is 0.907 (p <0.01).

Nevertheless, there are still some inconsistencies between them. For example, a developer has higher H_{times} while his H_{loc} is comparatively smaller. This implies that, the distribution of LoC in his active behaviors is more unbalanced and



(a) Developer 3 of JUnit.



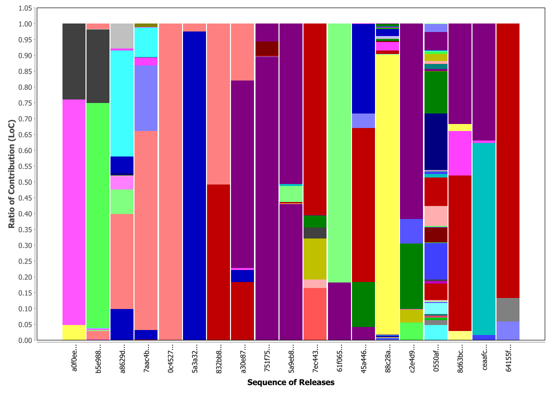
(b) Developer 4 of Guava.

Figure 3. Distribution of intervals between neighboring contributing behaviors

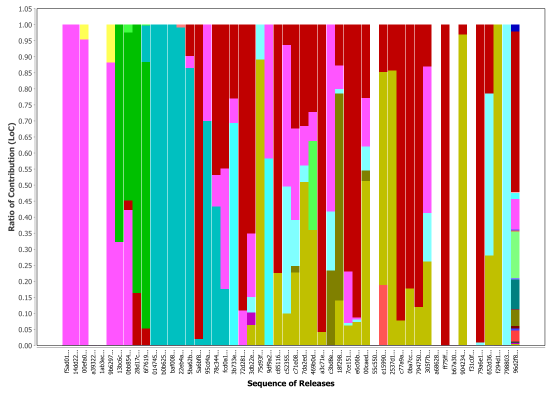
changes more drastically than the distribution of behavior times. For example, there is a developer in `JUnit` with almost all his LoC (nearly 45 KLoC) contributed in the last period of his participation, and the LoC in other periods are very small, thus his H_{times} is in rank 24 among all developers while his H_{loc} is in rank 70 only.

We observe from the barcode that, contributions of many developers are distributed in a stage-wise manner, i.e., between two phases of intensive active behaviors, there is usually a long interval. Fig. 3 shows $V_{interval}$ of two developers in `JUnit` and `Guava`, respectively. Although a majority of the intervals are close to x-axis (indicating the intervals are short and the contributions are approximately consecutive), there are obviously some outliers that have relatively higher values and represent the long gaps between stages of behaviors. For example, the developer in Fig. 3(a) has two obvious intervals, the first one is one year, and the second is two years. By contract, although Fig. 3(b) seems to have more frequent fluctuations, the amplitudes of these fluctuations are actually smaller than the ones of the first developer (see the different scale of y-axis in the two figures).

Summary: (1) Barcode is an intuitive method for visializing temporal distribution of a developer's contributions, and entropy can measure the continuity of contribution distribution; (2) There are temporal locality in the contributions of developers, i.e., most of the intervals between neighboring behaviors are short; but there are a small number of long intervals that split the behavior sequence into stages.



(a) JUnit.



(b) Guava.

Figure 4. Stacked barchart of the relative contribution ratio of developers in releases

3.2. Distribution of Contributions w.r.t. Releases

$V(r_k) = \langle \sigma_{1k}, \sigma_{2k}, ..., \sigma_{\tau k} \rangle$, vector of the contribution ratios of developers in one release period r_k, is visualized first. In the form of stacked barchart in Fig. 4, all the releases appear side by side for comparison. The x-axis is a set of releases sorted in chronological order, and the y-axis is the ratio of contributions. Sub-bars with the same color in different releases represent the same developer.

Some significant facts are observed: (1) There are usually a few developers whose contributions dominate one release. The number of dominant developers is so few that the contributions of other developers involving in the same release eventually become less obvious. (2) A small number of releases possess a large number of developers (e.g., 4th release from the end in `JUnit` and the last release in `Guava`). (3) Significant contributions that a developer makes are generally located in one single or multiple consecutive releases.

We use 5% as the threshold of judging whether a developer is a “significant contributor” of a release, i.e., if the ratio of the added LoC of a developer relative to the total added LoC of a release is above 5%, he is considered to be a significant developer of this release. Sensitivity analysis has shown that the value of this threshold ranging from 2% to 10% yields similar results.

In `JUnit`, the range of number of contributors in one release is $[1, 59]$, but most of releases have less than 10 developers, above 80% releases have less than 3 significant contributors, and the contribution ratio of the leading developer is usually far beyond the one of the runner-up. Similar situation can be found in `Guava`, too. Comparing the two projects we find that the average and median number of contributors of one release in `JUnit` are both larger than the ones in `Guava` (average: $9.42 > 3.30$; median: $5 > 2$), and the average variance of contribution ratios of these contributors in `JUnit` is smaller than the one of `Guava` ($0.025 < 0.04$). These statistical results show that the participation degree of developers in `JUnit` is higher than the one in `Guava`; in other words, developers in `JUnit` are more active than the ones in `Guava`.

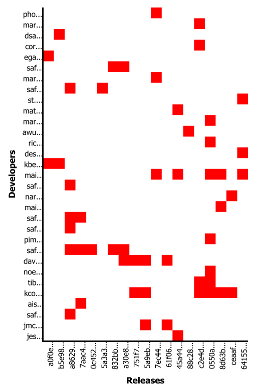
In the heat chart of Fig. 5, if a developer in y-axis is a significant contributor of a release in x-axis, the corresponding grid is filled with red color. Developers who are not significant contributors of any releases are not included in the figure. The heart chart further demonstrates the continuity (temporal locality) of the active behaviors of developers relative to the milestones of OSS projects, i.e., for most of developers, their active behaviors dominate one or several releases which are adjacent or close to each other.

To note that, the absolute timespan of different release periods are quite different (see the distance between blue vertical bars in Fig. 1), and the total LoC of different releases are greatly different, too; thus, even if the contributions of a developer dominate a release, it does not indicate that his absolute LoC contribution are large. Once again, what Fig. 4 shows are the ratio of contributions in each release period.

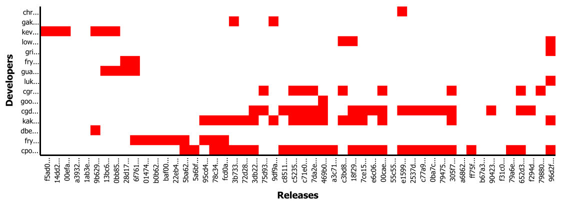
To sum up, the conclusions drawn in this section are: (1) Being the milestones of an OSS project, a majority of releases are dominated by very few developers whose active behaviors constitute the main code changes of the releases. The number of contributors of most releases is usually low, and only very few releases have a large number of contributors whose contributions are relatively balanced. (2) Contributions of a developer tend to be distributed in neighboring or near releases. This demonstrates the temporal locality of a developer’s active behaviors relatively to the staged milestones of the project. (3) The distribution of the number of dominating releases of all the developers exhibits approximate power law.

4. Threats to Validity

We select DVs in three levels, i.e., the temporal distribution of contributions for the most detailed level; the intensity and



(a) JUnit.



(b) Guava.

Figure 5. Heat chart of significant developers relative to releases

contribution ratio relative to project releases for the middle level; and the entropy for the global level. These DVs are calculated from the IVs which are derived from the open data of the OSS repositories hosted on Github. Such multi-level DVs ensure the comprehensiveness and understandability of the measurement.

Active behaviors are defined as the contributing activities that developers take on their own initiatives, mainly the committing activities that have explicit code contributions to projects. The exclusion of other types of activities (such as forking a project, commenting, or reporting an issue) from the study would result in some incompleteness of the identified individualized behavioral characteristics. Nevertheless, many existing research adopted similar approaches as ours.

Only the added lines of code in commits are considered but the “deletions” are ignored. However, if a developer removes redundant or buggy code without adding new lines, he still has contributions to the project. It makes the measurement on the continuity of contributions a little biased.

Two OSS projects are selected for the case studies. In an attempt to address the generalizability of our findings, more s-tudies conducted on larger-scale and longer-living OSS projects are necessary.

5. Related Work

Code commitment is considered as the dominant behaviors of developers because they take direct effect on the source code of OSS projects, e.g., Teyton *et al* [7] extracted a developer's contribution by delta analysis on source code changes being the result of developer behaviors. Yang *et al* [8] analyzed the commit activities of developers and found that they are not regularly distributed along with time, thus "barcode" is used to visualize the distribution of commit sequence of each developer. Our study extends their barcode by (1) using the height instead of the width of bars to signify the attributes (times, LoC, and intensity) of behaviors, and (2) putting the barcode into a timeline, so that the temporal distribution of contributions is visualized more precisely.

Siy *et al* [9] presented a method to summarize work history of developers in terms of the files they have modified over time using the time series segmentation technique, and the segmentation result is used as an individualized feature of developers and some evolution patterns of developer behaviors are found. In our study, we use two types of time segmentation approaches to split the developer behaviors into segments, i.e., absolute and relative timelines.

Lin *et al* [10] studied the relationship between the distribution of commit behaviors and the releases of OSS projects and identified five distinct zones in the distribution of commit activities across various releases; their conclusion indicates that developer behaviors are not independent but partially influenced by global constraints of projects such as "deadline". In our study, we borrow this idea and use releases as the relative timeline to study the correlation between developers' contributions and project milestones.

Weissgerber *et al* [11] used a transaction visualization technique to show commit sequences in a project and used file-author matrix to visualize evolution history of a file in terms of developers. This approach stands on the file's point of view; by contrast, we use a line of code as basic change unit rather than a large-grained file, thus the result would be more accurate.

6. Conclusion

We conduct an case study on the individualized features and common patterns of contributions of OSS developers. Continuity of active contributing behaviors is focused to explore how developer behaviors are distributed over time by a set of fine-grained metrics. The conclusions are: (1) Active behaviors of a developer often show temporal locality. Some long intervals split a developer's behaviors into stages, and the more contributions a developer makes, the more evenly his active behaviors are distributed over time, with averagely shorter intervals between neighboring stages of intensive active behaviors. (2) Most of releases of a project are dominated by limited number of developers, respectively, and the total number of contributors in each release is usually low. Active behaviors of a developer tend to be distributed in neighboring or near releases.

The findings of this study would help OSS project coordinators get deep insight in the behavioral characteristics of team members so as to improve their project management practices. Future work will be conducted on this perspective.

Acknowledgment

Work in this paper is supported by the Natural Science Foundation of China (No. 61272187, 61472106).

References

[1] J. A. Roberts, I.-H. Hann, and S. A. Slaughter, "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects," *Management Science*, vol. 52, no. 7, pp. 984–999, 2006.

[2] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research Policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[3] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, "Who is an open source software developer?" *Comm. the ACM*, vol. 45, no. 2, pp. 67–72, 2002.

[4] M. Y. Allaho and W.-C. Lee, "Trends and behavior of developers in open collaborative software projects," in *2014 Int'l Conf. Behavior, Economic and Social Computing*. IEEE, 2014, pp. 1–7.

[5] W. Harrison, "An entropy-based measure of software complexity," *IEEE Trans. Software Engineering*, vol. 18, no. 11, pp. 1025–1029, 1992.

[6] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta, "How changes affect software entropy: an empirical study," *Empirical Software Engineering*, vol. 19, no. 1, pp. 1–38, 2014.

[7] C. Teyton, M. Palyart, J.-R. Falleri, F. Morandat, and X. Blanc, "Automatic extraction of developer expertise," in *18th Int'l Conf. Evaluation and Assessment in Software Engineering*. ACM, 2014, p. 8.

[8] W. Yang, B. Shen, and B. Xu, "Mining github: Why commit stops–exploring the relationship between developer's commit pattern and file version evolution," in *APSEC'13*. IEEE, 2013, pp. 165–169.

[9] H. Siy, P. Chundi, and M. Subramaniam, "Summarizing developer work history using time series segmentation: challenge report," in *MSR'08*. ACM, 2008, pp. 137–140.

[10] S. Lin, Y. Ma, and J. Chen, "Empirical evidence on developer's commit activity for open-source software projects," in *SEKE'13*, 2013, pp. 455–460.

[11] P. Weissgerber, M. Pohl, and M. Burch, "Visual data mining in software archives to detect how developers work together," in *MSR'07*. IEEE, 2007, pp. 9–16.