Early Identification of Future Committers in Open Source Software Projects

Akinori Ihara*, Yasutaka Kamei†, Masao Ohira‡, Ahmed E. Hassan§, Naoyasu Ubayashi† and Ken-ichi Matsumoto*
*Graduate School of Information Science, Nara Institute of Science and Technology, JAPAN
Email: (akinori-i, matumoto)@is.naist.jp
†Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, JAPAN
Email: kamei@ait.kyushu-u.ac.jp, ubayashi@acm.org
‡Faculty of Systems Engineering, Wakayama University, JAPAN
Email: masao@sys.wakayama-u.ac.jp
§School of Computing, Queen's University, CANADA
Email: ahmed@cs.queensu.ca

***Abstract*—There exists two types of developers in Open Source Software (OSS) projects: 1) Committers who have permission to commit edited source code to the Version Control System (VCS), 2) Developers who contribute source code but cannot commit to the VCS directly. In order to develop and evolve high quality OSS, projects are always in search of new committers. OSS projects often promote strong developers to become committers. When existing committers find strong developers, they propose their promotion to a committer role. Delaying the committer-promotion might lead to strong developers departing from an OSS project and the project losing them. However early committer-promotion comes with its own slew of risks as well (e.g., the promotion of inexperienced developers). Hence, committer-promotion decisions are critical for the quality and successful evolution of OSS projects. In this paper, we examine the committer-promotion phenomena for two OSS projects (Eclipse and Firefox). We find that the amount of activities by future committers was higher than the amount of activities by developers who did not become committers). We also find that some developers are promoted to a committer role very rapidly (within a few month) while some of developers take over one year to become a committer. Finally, we develop a committer-identification model to assist OSS projects identifying future committers.**

I. Introduction

In an Open Source Software (OSS) project, some developers (called "committers") have permission to commit edited source code (e.g., patches) to the Version Control System (VCS) of their project [1]. An OSS project gives such permission to a very limited number of developers. The small number of committers ensures that OSS projects can maintain the high quality of their source code.

However as the popularity of an OSS project grows, the number of contributed patches increases at a very fast pace. Such rapid growth of contributed patches makes it difficult for a relatively few committers to manage them in a timely fashion [2][3][4][5].

Large-scale successful OSS projects need to increase the number of committers to successfully evolve [6]. However, most large-scale OSS projects have more than ten thousand developers [7]. It is difficult to easily identify a potential committer (i.e., a developer that has good promise to become a committer). In general, a developer is promoted to a new committer through the recommendation of current co mmitters [8]. The committers comprehensively appraise developer's activities in extending functions and fixing bugs. However, potential future committers often leave the project before they are recommended to a committer role [9][10].

To avoid the loss of capable developers, one needs to identify *future committers* as soon as possible. However, the committers do not know whether a developer recommended by committers is likely to continue to contribute to the project for a long time. In this study, to identify future committers, we analyze developers activities in terms of the number of contributed patches and the number of contributed comments. Then we build a committer-identification model using these metrics. The goal of the model is to predict whether a particular developer will became a committer during our target period.

Using data from two large OSS projects (Eclipse platform and Mozilla Firefox), we answer three research questions.

RQ1: Are there any differences in the activities of future committers and developers?

To understand the developer activities that contribute to the predictive accuracy of a committer-identification model, we analyze the differences in the activities of future committers and developers (i.e., not future committers). If we find any differences of activities, these metrics would present an effective measure in identifying future committers.

RQ2: Which developer activities lead to early promotion to a committer role?

Most committers are promoted to such a role after contributing to the project for around a year. However some developers are promoted to a committer role considerably earlier. Hence, we analyze the activities of *rapidly-promoted* committers, and the activities of regularly-promoted committers.

RQ3: How accurate is a committer-identification model built using developer activities?

We build a committer-identification model using developer activities (i.e. patch submission, discussion of development, and activity period), then we evaluate how well the model predicts future committers.

This paper is laid out as follows. Section 2 introduces related work motivates our study. Section 3 presents the studied developer activities. Section 4 provides the design of our experiment, and Section 5 presents the results. Section 6 studies the activities of developers once they become committer. Finally, Section 7 concludes the paper and presents our future work.

II. Background and Related work

A. Support for the activities of committers

Many researchers have proposed approaches to support committer activities [11][12][13]. A prominent goal of many of these studies has been to automate some of the committer activities [14][15][16].

Triagging Bugs: Committers receive a large number of feature requests and bug reports. Rastkar et al. [16] built a system to automatically summarize bug reports in order to reduce the time needed to understand these bug reports. Moreover, Hooimeijer et al. [17] presented a method to identify the bug reports that should be fixed first using metrics derived from developer activities data such as the number of comments.

Assigning bugs: When committers receive requests for fixing a bug, they should assign the requests to developers who have the appropriate skills to solve the request. If the assignment is done incorrectly, valuable resources are wasted [18]. Anvik et al. [14] and Cubranic et al. [15] presented a approach to identify the most suitable developer for a bug report based on textual information such as the title and the description of the bug report.

Verifying bugs: Committers have to verify patches that developers created to fix a bug. After that, committers commit the fixed source code to the VCS of the project. Unfortunately in some cases, bugs have to be re-opened [19]. Re-opened bugs increase maintenance costs and lead to unnecessary rework by busy developers. Shihab et al. [20] presented an approach to identify whether a bug will be re-opened.

Prior studies aim to provide techniques for the effective use of the limited human resources of a few committers. On the other hand, the goal of our study is to increase the human resources by recommending additional capable committers.

B. Studies of the Committer Promotion Process

Increasing the number of committers in a large OSS projects is important to cope with the large influx of developer contributions and requests [1]. In order to identify future committers from developers, current committers carefully examine developer activities in search of future committers that can join the project.

Jensen et al. [8] interviewed committers about the activities that they track to recommend new committers. In the Apache project, committers identify new committers based on the developer's technical activities such as patch contributions. The recommended future committers are presented to Apache PMC (Project Management Committees) members. The PMC members judge whether or not a proposed committer should be promoted to a committer role. In the Mozilla project, committers identify future new committers based on a developer's social activities such as discussion of OSS development in addition to their technical activities.

Zhou and Mockus [21] analyzed how the expertise of developers increases in software projects. They found that developers' productivity in terms of the number of tasks per month increases with project tenure and plateaus within a few months in small and medium projects, while taking up to 12 months in large projects. In an extended study, they analyzed what impacts the chances that a new joiner to a software project will become an LTC (Long Term Contributor) who stays with the project for at least three years [22]. As a result, they found that the main differences among participants were in their capacity, willingness and opportunity to contribute to activities (e.g. the number and type of tasks, the fraction of reported issues) at the time of joining. Moreover Zhou et al. did target not only committers but also all developers in general.

Bird et al. [9] presented a hazard-rate model to identify the most active period of developers in the Apache project, Python project and PostgreSQL project. They found that existing committers usually recommend developers who have worked for about a year. In addition, prior history of patch submissions has a strong effect in Apache and Python projects. However, Bird et al. have not explored the activities of future committers before they become committer (or after they become committers). Moreover, Bird et al. have not examine the difference of activities between regular and rapidly-promoted committers.

Fujita et al. [23] analyzed the differences in the number of submitted and reviewed patches, and the patch edit and review times between future committers and developers. As a result, they found that future committers contributed technical contribution (patch submission and review) more than developers. However, they have not analyzed differences of social activities (i.e. communication among developers). In addition, Fujita et al. did not show the change in activities as developer are promoted to committer roles. Moreover, they did not explore the various types of committer promotions.

Gharehyazie et al. [24] built statistical predictors for future developers (like commimtter) in OSS projects based on activities (the number of patches/messages) early in their tenure with the project. As a result, developer initiation could be modeled with as little as one month's work of information about the social activity of individuals. When Gharehyazie et al. used the information for three months since an individual started, the model produced more stable result. Moreover, in order to identify high potential future committers, they did not focus on the differences of their promotion process and their activities after being a committer.

III. DEVELOPERS ACTIVITY

Our studied OSS projects (Eclipse and Mozilla) provide guidelines for committer promotions[1]. According to the guidelines, when the project acknowledges the contribution of a developer, he or she will be promoted to a committer role.

[1]Eclipse: http://wiki.eclipse.org/Development_Resources/HOWTO/Nominating_and_Electing_a_New_Committer
Mozilla: http://www.mozilla.org/hacking/committer/

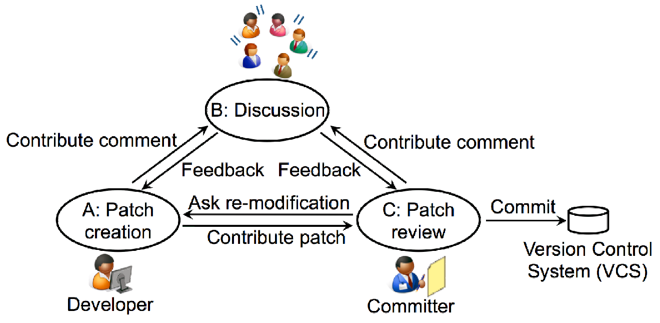

Fig. 1: Process for applying patches in a project.

However, the guidelines do not define the specific activities or the amount of activity to be used to appraise the developer's contribution. In this study, we compare the activities of the future committers with the activities of developers who did not become a committer during our study period. Figure 1 shows a simplified process for patch development.

A: Patch creation: Developers create patches to extend functionality and to fix bugs. Then, they contribute patches to a Bug Tracking System (BTS). In this study, we regard the developer who contributed a patch as the developer who created the patch.

B: Discussion: Developers discuss the creation of patches and review each other's patches. They discuss their plans and designs.

C: Patch verification: Committers verify the contributed patches. If they judge that a patch should be edited again, they would ask the developers to re-edit the source code. On the other hand, if they judge that a patch does not need to be edited again, the committers would commit the patch to the Version Control System (VCS) on behalf of the developers.

Only committers can verify patches (C). We analyze the developer activities (A: Patch creation and B: Discussion) and the time period which committers use to appraise the activities of a future committer. We then build a committer-identification model using these activities.

IV. Experiment Setting

To understand the developer activities that existing committers appraise when identifying future committers, we analyze the activities of future committers before they become committers.

A. Target Data

Our study uses data from the Eclipse and Mozilla projects. Table I presents the studied period, and the number of developers in each project.

Some developers are already committers. For example, some IBM company developers have been committers since they joined the Eclipse project. *Existing committer* are ones who have committed source code to VCS before the studied period (Figure 2. On the other hand, we describe a developer as a *future committer*, he or she commits patches for the first time during the studied period. We also describe someone as a "developer" if he or she has never committed anything to VCS since the project's start and throughout the studied period. We found 36 existing committers, 53 future committers and 8,964 developers in the Eclipse platform project. We focus on only 9,017 (= 53 + 8,964) developers. We do not consider the 36 existing committers, since the goal of this study is to develop a method for identify future committers.

TABLE I: Summary of the studied data.

	Eclipse project	Mozilla project
subproject	platform	Firefox
target period	2001/10-2010/12	2004/01-2008/12
existing committers	36	96
developers	8,964	12,287
future committers	53	51
studied developers (developers + future committers)	9,017	12,338

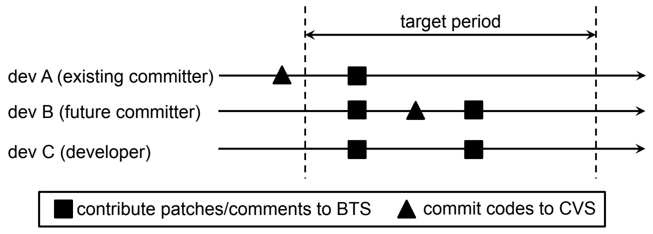

Fig. 2: An example of classifying developers.

B. Extracting Metrics

In this study, we extracted metrics from the Bugzilla[2] data as BTS about (a) Patch creation and (b) Discussion shown in Figure 1, and extracted the committer list from VCS [3]. Figure 3 shows the procedure used to extract metrics using the BTS and VCS data.

(1) Extracting developer activities from BTS data

Many OSS projects use a BTS to manage submitted enhancement requests and bug reports. BTS reporters write down the target module name, attach edited patches, and submit comments in the reports. In our study, we collected the reports from BTS, then extracted who contributed the patches/comments, when developers/users contributed them. This automated extraction method is similar to prior studies [23][25][26].

We also measure the activity period of each developer. The activity period is from the month when a developer started their first activity (patch or comment contribution) in a project to the month when they performed their final activity during our studied period. We summarize their activity history in developers activities reports for each developer ((a) in Figure 3).

[2]Eclipse Bugzilla: https://bugs.eclipse.org/bugs/
Mozilla Bugzilla: https://bugzilla.mozilla.org/

[3]Eclipse and Mozilla VCS were provided by the MSR Challenge conference (http://2011.msrconf.org/)

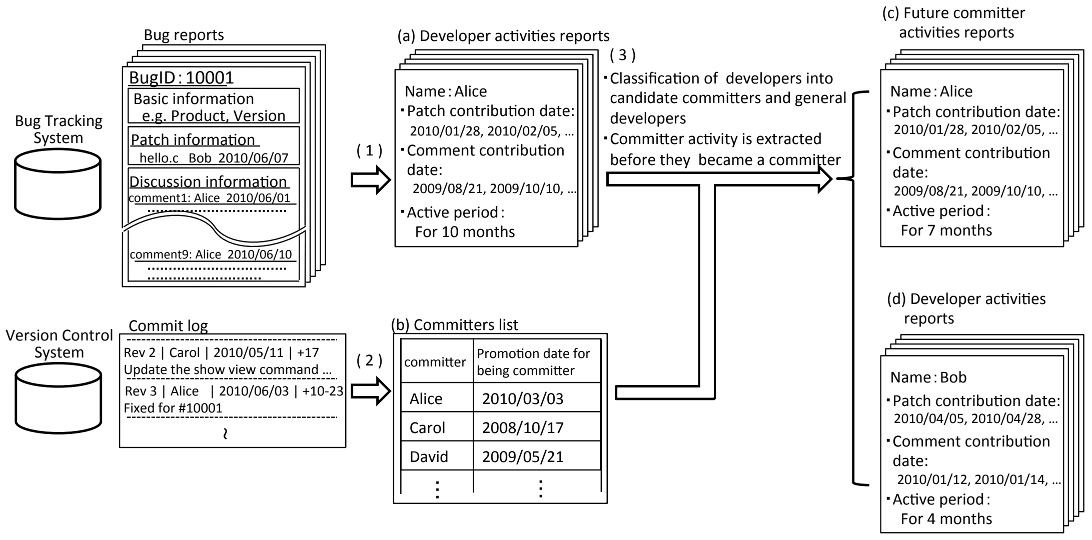
Bug reports
Bug Tracking System
BugID : 10001
Basic information e.g. Product, Version
Patch information hello.c Bob 2010/06/07
Discussion information
comment1: Alice 2010/06/01
comment9: Alice 2010/06/10
(1)
(a) Developer activities reports
Name : Alice
・Patch contribution date: 2010/01/28, 2010/02/05, ...
・Comment contribution date: 2009/08/21, 2009/10/10, ...
・Active period : For 10 months
(3)
・Classification of developers into candidate committers and general developers
・Committer activity is extracted before they became a committer
(c) Future committer activities reports
Name : Alice
・Patch contribution date: 2010/01/28, 2010/02/05, ...
・Comment contribution date: 2009/08/21, 2009/10/10, ...
・Active period : For 7 months
Version Control System
Commit log
Rev 2 | Carol | 2010/05/11 | +17
Update the show view command ...
Rev 3 | Alice | 2010/06/03 | +10-23
Fixed for #10001
(2)
(b) Committers list
committer
Promotion date for being committer
Alice 2010/03/03
Carol 2008/10/17
David 2009/05/21
(d) Developer activities reports
Name : Bob
・Patch contribution date: 2010/04/05, 2010/04/28, ...
・Comment contribution date: 2010/01/12, 2010/01/14, ...
・Active period : For 4 months

Fig. 3: Method to extract developer activities.

(2) Extracting the promotion date for becoming a committer from VCS data

We collect the commit log from VCS, then we summarize the committer name and the date when each committer committed their first patch to VCS. We record each committer and the date of their first patch (i.e., the date they became a committer) in a Committers list ((b) in Figure 3).

(3) Classifying the developer activities reports

Using the committer list, we can classify developer activities reports into either future committer activities reports or developer activities reports ((c) and (d) in Figure 3). The future committer activities tracks activities of a developer until the developer is promoted to a committer.

V. Experimental Results

The goal of our experiment is to understand the committer activities and the accuracy of our committer-identification model. We now present the results of our study with respect to our three research questions.

RQ1: Are there any differences in the activities of future committers and developers?

Motivation. As shown in Figure 1, OSS developers iterate patch creation and discussion during development to enhance functionalities and to fix bugs. After that, committers verify contributed patches and commit the patches to VCS. Committers recommend a high potential developer as a new committer (i.e., future committer). Then existing committers appraise the developer's past activities (patch and discussion contributions). There may be differences in the amount of activities between a future committer and a developer. If we can find such differences, we can easily identify future committers from the thousands of developers in an OSS project. For RQ1, we compare the amount of activities of future committers and developers to develop an effective mean to identify future committers.

Approach. We present two steps in this approach. These steps are (1) Extracting future committer activities and developer activities, and (2) Testing the differences in the activities of future committers and the activities of developers.

(1) Extracting future committers activities and developers activities

Existing committers appraise a developer's past activities (patch and discussion contributions) to find future committers. In this study, we measure the number of patch files/comments that developer contributed to Bugzilla. The activity period of a future committer is up till he or she commits patches to VCS for the first time. The activity period of a developer is all his activities during our studied period.

(2) Testing the differences between the activities of future committers and the activities of developers

We conducted a Wilcoxon signed-rank test. A non-parametric tests which does not assume a normal distribution as is the case in our data set. We used a significance level of 5%.

Results. Figure 4 and 5 present the differences in the amount of activities between future committers and developers. Figure 4 shows the distribution of the number of contributed patches in the Eclipse platform and Mozilla Firefox projects by future committers and by developers. Figure 5 shows the distribution of the number of contributed comments by future committers and by developers. The p-value of the Wilcoxon signed-rank test are noted at the top of each sub-figure.

The amount of activities of future committers is higher than that of developers. In the Eclipse project, the median number of contributed patches by future committers and by developers is 15 and 0 respectively. In the Mozilla Firefox project, the median number of contributed patches by future committers and by developers is 1 and 0 respectively. In both projects, we find a significant difference in the number of contributed patches by future committers and by developers. Hence, most

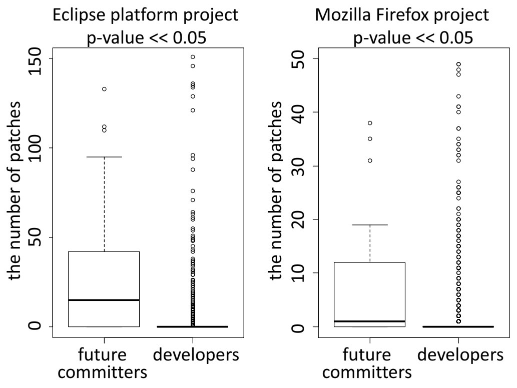

Fig. 4: Differences of the number of patches between committers and developers

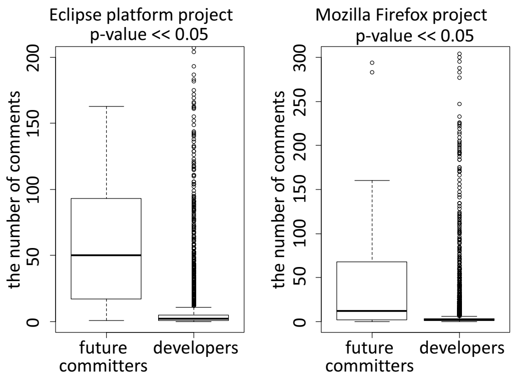

Fig. 5: Differences of the number of comments between future committers and developers

developers have never contributed patches. Also, in the Eclipse platform project, the median number of contributed comments by future committers and by developers is 50 and 2 respectively. In the Mozilla Firefox project, the median number of contributed comments by future committers and by developers are 12 and 2 respectively. In both projects, we also find a significant difference in the number of contributed comments by future committers and those by developers. Most developers also did not contribute social activities. Hence both metrics (the number of contributed patches/comments) are useful in identifying future committers.

RQ1: The amount of activities by future committers is higher than the amount of activities by developers. These metrics (the number of submitted patches/comments) are useful in identifying future committers.

RQ2: Which developer activities lead to early promotion to a committer role?

Motivation. Committers usually consider a developer for a committer role once the developer has worked for about a year [9]. However, some developers who have worked for less than a year have become committers. We compared the number of committers who has contributed for more than one year or less.

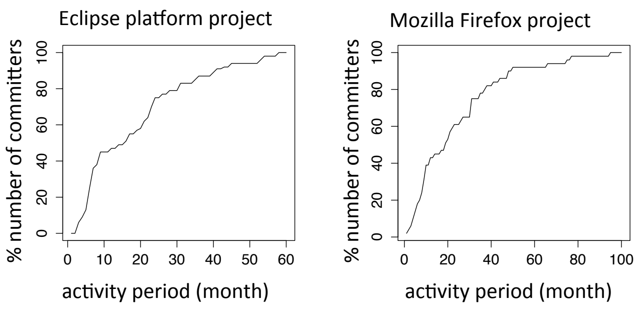

Fig. 6: Activity period until becoming a committer.

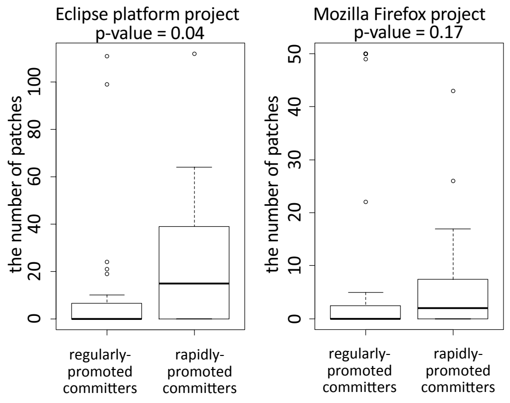

Fig. 7: Differences of the number of patches of regularly-promoted committers and rapidly-promoted committers

Figure 6 shows the pareto chart of the activity period before becoming a committer in studied projects (the Eclipse platform and Mozilla Firefox projects). In Eclipse project, we found 25 regularly-promoted committers and 28 rapidly-promoted committers who have worked for more than (less than) a year. Similarly, in Mozilla Firefox project, we found 29 regular experienced committers and the 22 rapidly-promoted committers.

For RQ2, we analyze the activity patterns of regularly-promoted committers and those of rapidly-promoted committers. We break up RQ2 into two sub-research questions.

RQ2-1: Is there a difference in activities of rapidly-promoted committers and the activities of regularly-promoted committers?

Approach. We compare the rapidly-promoted committer activities to the regularly-promoted committer activities. The rapidly-promoted committer activities is the amount of activities (patch and discussion contributions) performed by the rapidly-promoted committers. The regularly-promoted committer activities is the amount of activities performed by the regularly-promoted committers. We conducted a Wilcoxon signed-rank test on their activities.

Results. Figure 7 and 8 show the Pareto chart distribution of

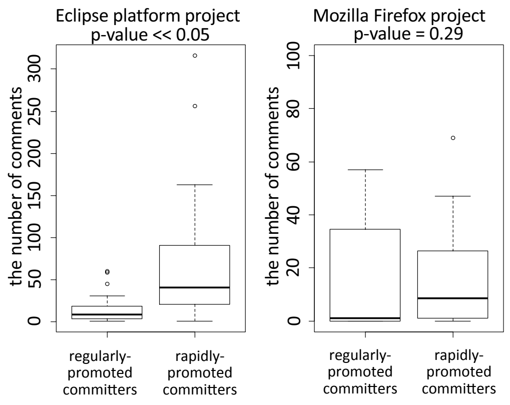

Fig. 8: Differences of the number of comments of regularly-promoted committers and rapidly-promoted committers

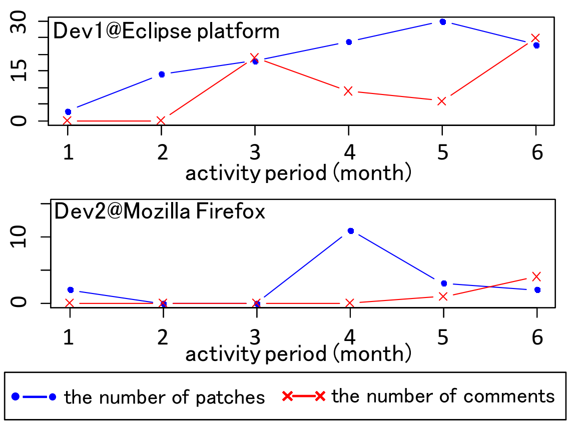

Fig. 9: Activity change of rapidly-promoted committers.

the activities (the number of contributed patches/comments) of regular and rapidly-promoted committer. The p-value of the Wilcoxon signed-rank test are noted at the top of the sub figures.

First, we compare the number of contributed patches. In the Eclipse project, the median number of contributed patches by regularly-promoted committers is 0 and by rapidly-promoted committers is 15. In the Mozilla Firefox project, the median number of contributed patches by regularly-promoted committers is 0 and by rapidly-promoted committers is 2. In summary, the median number of contributed patches by regularly-promoted committers is less than the number of contributed patches by the rapidly-promoted committers.

Next, we compare the number of contributed comments. In the Eclipse platform project, the median number of contributed comments by regularly-promoted committers is 9 and by rapidly-promoted committers is 41. In the Mozilla Firefox project, the median number of contributed comments by regularly-promoted committers is 1 and by rapidly-promoted committers is 11. Hence, that the median number of contributed comments by regularly-promoted committers is less than the number of contributed comments by the rapidly-promoted committers.

However, we could not find a significant difference in the number of activities between the regularly-promoted committers and the rapidly-promoted committers in the Mozilla Firefox project.

Figure 9 shows the activity change of rapidly-promoted committers. We analyze a period after these future committers have started working in the projects. The figure shows that Dev 1 and 2 are promoted to committers after only 6 months from starting to contribute to the project. As a result of these activities, the rapidly-promoted committers posted at least over 10 times comments or patches in a month in their early days as committers.

RQ2-1: The number of activities performed by rapidly-promoted committers is more than regularly-promoted committers.

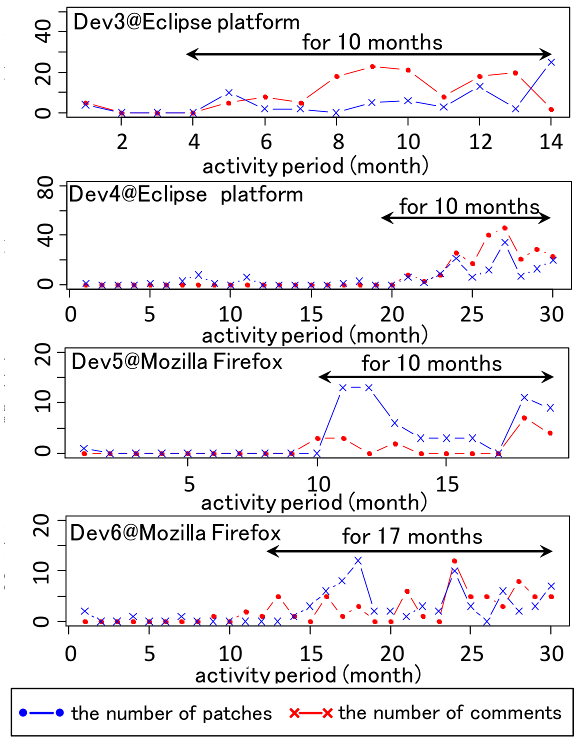

Fig. 10: Activity change of regularly-promoted committers.

RQ2-2: What do regularly-promoted committers do more than the rapidly-promoted committers?

Approach. For RQ2-2, we qualitatively analyzed the change activities (the number of contributed patches/comments) in each month until a regularly-promoted committer becomes a committer.

Results. Figure 10 shows the activity changes of regularly-promoted committers in each project. Looking at these figures, we see a difference between the rapidly-promoted committers and the regularly-promoted committers. The amount of activities by regularly-promoted committers (such as Dev3, 4, 5 and 6 in Figure 10) is fewer than the rapidly-promoted

TABLE II: Developer activities.

activities	metrics	detail
Patch creation	SumNumPatch	Sum of the patches that a developer submitted to BTS.
	MedNumPatch	Median of the patches that a developer submitted to BTS.
Discussion	SumNumComment	Sum of the comments that a developer submitted to BTS.
	MedNumComment	Median of the patches that a developer submitted to BTS.
Activity Period	ActPeriod	Period that a developer has worked in the OSS project.

committers (such as Dev1,2 in Figure 9) at the beginning of their time. However, the regularly-promoted committers gradually increase the amount of their activities, and they became committers after they have worked actively for 1-1.5 years. Few developers continue actively work for about a year. Most developers who have actively worked for about 1-1.5 years are promoted to committer role.

It is interesting to note that if one examines the few months before committer-promotion, we find that the amount of activity of rapidly-promoted committers is similar to the amount of activity of regularly-promoted committers. Hence it appears that the amount of activity plays a much bigger role than simply the amount of time spent as a developer – this observation might help explain the commonly-discussed in literature one-year holding period before promotion (e.g., [9]) – It is most likely that developers take one year to ramp up their contribution levels. However, if a developer is able to ramp up their contribution faster than there is a good chance that the developer will be promoted to a committer role much faster (as is the case in over 40% of the promoted committers in both projects).

RQ2-2: The amount of contribution in the time period just pre-promotion plays a big in the committer promotion.

RQ3: How accurate is a committer-identification model built using developer activities?

Motivation. Current committers have to find capable OSS developers from the thousands of developers in an OSS project in order to promote these capable developer to committer roles. Current OSS committers would be benefit from a prediction model that can automatically identify capable developers that are likely to become committers.

Approach. We build a committer-identification model using the developer activities (patch and discussion contributions) and the activity period. From RQ2-1, we found that the rapidly-promoted committers contributed many patches and comments in a short time. From RQ2-2, we find that most developers who have actively worked on a project for 1-1.5 years are likely to become a committer.

In this study, we use the sum of the activity and the median of the activity to build the model. Table II shows the developer activity metrics that are used to build the model.

Our model uses the random forest algorithm to identify future committers [27]. The objective variable is whether or not a developer became a committer during our studied period (i.e., a future committer or not). The model identifies a developer as a future committer when the output (continuous value: 0-1) of the model is over a threshold (0.2, 0.5, 0.8).

We randomly divide our committer dataset (Fig.3 (c)) and developer dataset (Fig.3 (d)) into two sets: one for training and the other for testing. However, the number of developers is much more than the number of committers (9,017:53 for Eclipse and 12338:51 for Mozilla). With over 240:1 imbalance, special attention is needed for the model building and for the model evaluation [28][29]. In particular, we need to rebalance the training data (we do not modify the testing data) – we sampled the same number of developers as committers to build the model. We also rebuild the model 100 times and use the median of the 100 evaluation results as the experimental result. Given the high imbalance characteristics of our dataset, it is recommended to examine the AUC (Area Under the Curve) of ROC (Receiver Operating Characteristic) instead of the classically-used precision, recall and F1 metrics in most software engineering studies [28][30][31]. Nevertheless we do show all evaluation metrics not only the AUC metric.

Precision measures the ratio of the number of developers who actually became committers to the number of predicted committers. Recall measures the ratio of the number of developers who became actual committers to the number of predicted developers who never became committers. F1-value is a combined value of recall and precision as follows.

$$F1{-}measure = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

These criteria (e.g., precision and recall) depend on the particular threshold used for the classification. We additionally use the AUC of ROC. The ROC measures the ratio of the number of developers who became committers to the number of the developers who never became committers. We evaluate AUC in terms of the number of developers who never became committers. The AUC metric ranges from 0 to 1. An AUC of 0.5 indicates that a model is equivalent to random guessing.

Results. Table III and table IV show the evaluation results. These table show that a very strong AUC of the model using all activities. The AUC is 0.94 and 0.95 for Eclipse platform and for Mozilla Firefox project. We find that the model had higher accuracy than the random predictor since the AUC is considerably higher than 0.5 in both projects. We also find that precision and recall using all metrics are about 18% and 58% in the Eclipse platform project (at 0.80 threshold), and about 8% and 44% in Mozilla Firefox project (at 0.8 threshold as well).

Next, we examine the factors which influence the random forest predictor the most (this is done by examining the variable importance measure that is computed by the random

TABLE III: Committer prediction results in Eclipse platform project.

	Threshold	Precision	Recall	F1-value	AUC
patch +comment +period	0.20	0.02	0.96	0.04	0.94
	0.50	0.05	0.77	0.09	0.94
	0.80	0.18	0.58	0.28	0.94
patch	0.20	0.01	1.00	0.01	0.81
	0.50	0.09	0.65	0.15	0.81
	0.80	0.14	0.54	0.22	0.82
comment	0.20	0.02	0.85	0.04	0.88
	0.50	0.04	0.73	0.08	0.89
	0.80	0.07	0.54	0.12	0.88

TABLE IV: Committer prediction results in Mozilla Firefox project.

	Threshold	Precision	Recall	F1-value	AUC
patch +comment +period	0.20	0.01	1.00	0.02	0.95
	0.50	0.02	0.96	0.04	0.95
	0.80	0.08	0.44	0.13	0.94
patch	0.20	0.00	1.00	0.01	0.76
	0.50	0.06	0.56	0.11	0.76
	0.80	0.06	0.56	0.11	0.76
comment	0.20	0.00	0.92	0.01	0.76
	0.50	0.01	0.72	0.03	0.76
	0.80	0.05	0.44	0.09	0.76

TABLE V: Variable importance measure.

	Eclipse project	Mozilla project
SumNumPatch	1.93	1.08
MedNumPatch	1.43	0.78
SumNumComment	3.76	1.38
MedNumComment	1.55	0.76
ActPeriod	2.29	3.47

forest). Table V shows the median value of each variable. From this table, we find that SumNumComment and ActPeriod are the most important factors in identifying future committers in both projects. This observation confirms prior findings by Bird et al. [9] about the importance of the length of developer involvement. However, we also find that social contributions (i.e., comment contributions) not just technical contributions (i.e., patch contributions) play a role in the promotion decision – the role of social contributions (3.76/1.38) is even larger than technical contributions (1.93/1.08) in both projects. Moreover, the role of social contributions (3.76) is strong even relative to activity period (2.29) in the Eclipse project.

RQ3: Our committer-identification model considerably outperforms a random predictor.

VI. Discussion

A. Activities after becoming a Committer

We have not confirmed whether or not future committers actively work after they become committers. Existing committers should recommend developers who will actively work after they become committers. To help address this, we analyzed the future committer activities after the future committer became a committer. In addition, we compared the regularly-promoted committer activities to the rapidly-promoted committer activities. The regularly-promoted committers may lose their motivation because they have already worked as developers for more than a year. Therefore, we analyzed what happens for new committers who have worked only for a short period of time before being promoted.

On the Eclipse platform project, we find that there are 40 committers of 53 total who have worked for more than a year after the future committer became a committer. On the Mozilla Firefox project, we found that there are 30 committers of 51 total who have worked for more than a year after the promotion.

Next, we analyzed the rapidly-promoted committer activities and the regularly-promoted committer activities after they became committers. Figure **??** and **??** show their activities. The result shows that the median number of comments and the median number of commits to VCS by the rapidly-promoted committers are higher than those by the regularly-promoted committers. We found significant differences in the distribution of the number of commits between the rapidly-promoted committers and the regularly-promoted committers (The significance level is 10%.).

Our results highlight the importance of early identification of strong developers and their rapid promotion to committer roles. In fact, rapidly-promoted committers actively contribute more than regularly-promoted committers.

B. Threats to Validity

We defined the date when a developer commits to VCS for the first time as the date of promotion to committer. However, in some rare cases a developer may get permission to commit patches before the date of promotion to committer.

We focused on developer activities in the patch contribution and integration process. There exists many other types of activities (such as the number of edited lines of code, the source code quality, and the content of comments) that could be studied in future work.

The number of the edited source code lines: Weißgerber et al. [26] show that the size of contributed patches is usually less than 10 lines. In our data, the number of contributed lines has a high correlation with the number of contributed patches. Therefore, we did not use the number of edited source code lines.

The source codes quality: Committers accept many contributed patches submitted by some developers. On the other hand, committers may reject many patches submitted by another developer. The acceptance or rejection rate could be an important metric to improve our model. However, it is difficult to know whether or not contributed patches were accepted, because linking the contributed patches to the committed source codes in VCS is not straightforward [32]. Currently, many researchers are trying to resolve this problem [33][34]. Future work should attempt to integrate such knowledge into committer-identification model.

The content of comments: Some developers contribute good comments to BTS. It might be wise to promote such developers to committers. However, measuring the quality of a contributed comment is very subjective, context-sensitive and not straightforward

In this paper, we targeted two large OSS projects, the Eclipse platform and Mozilla Firefox projects, according to the three conditions. (1) We can compare the results in each OSS project. We targeted OSS projects using Bugzilla as BTS for the condition. (2) The OSS projects are active and regularly release new versions. (3) There is enough data such as the number of commits and number of bug reports to preserve generality. We need to analyze other OSS projects to generalize our findings.

VII. Conclusions and Future work

In this study, we analyzed developer activities to identify future committers that should be promoted as committers in OSS projects. We built a committer-identification model to identify future committers among the thousands of developers in an OSS project. Using data from the Eclipse platform and Mozilla Firefox projects, we find:

RQ1 The amount of activity by a future committer is higher than the activity of a developer (i.e., not future committer).

RQ2 regularly-promoted committers have actively worked for 1-1.5 years before they become committers.

RQ3 Our committer-identification model outperforms a random predictor.

Furthermore, we found that after the promotion, rapidly-promoted committers were more active regularly-promoted committers.

In the future, we would like to enhance the committer-identification model by integration new metrics which capture other types of developer activities.

Acknowledgment

This work has been conducted as a part of "The Research Initiative on Advanced Software Engineering in 2013" supported by Software Reliability Enhancement Center (SEC), Information Technology Promotion Agency Japan (IPA). Also, part of this research was conducted under the Japan Society for the Promotion of Science, Grant-in-Aid for Young Scientists (B: 25730045), and Scientific Research (C: 24500041).

References

[1] K. Fogel, *Producing open source software: how to run sucessful free software project*. Sebastopol, CA: O'Reilly Media, 2005.

[2] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Proceedings of the 21st Symposium on Applied Computing (SAC'06)*, 2006, pp. 1767–1772.

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful... really?" in *Proceedings of the 24th International Conference on Software Maintenance (ICSM'08)*, 2008, pp. 337–345.

[4] A. Ihara, M. Ohira, and K. Matsumoto, "An analysis method for improving a bug modification process in open source software development," in *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops (IWPSE-Evol'09)*, 2009, pp. 135–144.

[5] M. Nurolahzade, S. M. Nasehi, S. H. Khandkar, and S. Rawal, "The role of patch review in software evolution: an analysis of the mozilla firefox," in *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops (IWPSE-Evol'09)*, 2009, pp. 9–18.

[6] B. Shibuya and T. Tamai, "Understanding the process of participating in open source communities," in *Proceedings of the ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS'09)*, 2009, pp. 1–6.

[7] V. S. Sinha, S. Mani, and S. Sinha, "Entering the circle of trust: Developer initiation as committers in open-source projects," in *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR'11)*, 2011, pp. 133–142.

[8] C. Jensen and W. Scacchi, "Role migration and advancement processes in OSSD projects: a comparative case study," in *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 364–374.

[9] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu, "Open borders? immigration in open source projects," in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, 2007, pp. 6–13.

[10] K. Yamashita, S. McIntosh, Y. Kamei, and N. Ubayashi, "Magnet or sticky? an oss project-by-project typology," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*, 2014, pp. 344–347.

[11] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles, "Supporting collaborative software development through the visualization of socio-technical dependencies," in *Proceedings of the International Conference on Supporting Group Work (GROUP'07)*, 2007, pp. 147–156.

[12] M. Lanza and M. Pinzger, ""A bug's life": visualizing a bug database," in *Proceedings of the 4th International Workshop on Visualizing Software for Analysis and Understanding (VISSOFT'07)*, 2007, pp. 113–120.

[13] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*, 2009, pp. 23–33.

[14] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, 2006, pp. 361–370.

[15] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE'04)*, 2004, pp. 92–97.

[16] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: a case study of bug reports," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE'10)*, 2010, pp. 505–514.

[17] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the 22nd International Conference on Automated Software Engineering (ASE'07)*, 2007, pp. 34–43.

[18] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the ESEC/FSE'09*, 2009, pp. 111–120.

[19] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, 2012, pp. 1074–1083.

[20] E. Shihab, A. Ihara, Y. Kame, W. M. Ibrahim, O. Ohira, Masao, B. Adams, A. E. Hassan, and K. Matsumoto, "Studying re-opened bugs in open source software," in *Empirical Software Engineering*, 2012, pp. 1–38.

[21] M. Zhou and A. Mockus, "Developer fluency: achieving true mastery in software projects," in *Proceedings of the 18th International Symposium on Foundations of Software Engineering (FSE'10)*, 2010, pp. 137–146.

[22] ——, "What make long term contributors: Willingness and opportunity in oss community," in *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, 2012, pp. 518–528.

[23] S. Fujita, A. Ihara, M. Ohira, and K. Matsumoto, "An analysis of committers toward improving the patch review process in OSS de-

velopment," in *Supplementary Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE'10)*, 2010, pp. 369–374.

[24] M. Gharehyazie, D. Posnett, and V. Filkov, "Social activities rival patch submission for prediction of developer initiation in oss projects," in *In Proceedings of the 29th International Conference on Software Maintenance (ICSM'13)*, 2013, pp. 340–349.

[25] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," in *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'10)*, 2010, pp. 301–310.

[26] P. Weißgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR'08)*, 2008, pp. 67–76.

[27] L. Breiman, "Random forests," in *Machine Learning*, vol. 45, no. 1, 2001, pp. 5–32.

[28] H. He and E. A. Garcia, "Learning from imbalanced data," vol. 21, no. 9, 2009, pp. 1263–1284.

[29] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," in *Technical Report666, Statistics Department, University of California at Berkeley*, 2004.

[30] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," in *ACM Transaction on Information Systems*, vol. 22, no. 49, 2004, pp. 5–53.

[31] S. Kim, E. J. Whitehead, Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?" in *IEEE Transaction on Software Engineering*, vol. 34, 2008, pp. 181–196.

[32] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the 18th International Symposium on Foundations of Software Engineering (FSE'10)*, 2010, pp. 97–106.

[33] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, 2005, pp. 1–5.

[34] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of Software Engineering (ESEC/FSE'11)*, 2011, pp. 15–25.