Who Will Become a Long-Term Contributor? A Prediction Model based on the Early Phase Behaviors

Tao Wang
Key Lab. of Parallel and Distributed Computing
Lab. of Software Engineering for Complex Systems
College of Computer, National University of Defence Technology
ChangSha, 410073, China
taowang2005@nudt.edu.cn

Yang Zhang
Key Lab. of Parallel and Distributed Computing
Lab. of Software Engineering for Complex Systems
College of Computer, National University of Defence Technology
ChangSha, 410073, China
yangzhang15@nudt.edu.cn

Gang Yin
Key Lab. of Parallel and Distributed Computing
Lab. of Software Engineering for Complex Systems
College of Computer, National University of Defence Technology
ChangSha, 410073, China
jack.nudt@nudt.edu.cn

Yue Yu
Key Lab. of Parallel and Distributed Computing
Lab. of Software Engineering for Complex Systems
College of Computer, National University of Defence Technology
ChangSha, 410073, China
yuyue@nudt.edu.cn

Huaimin Wang
Key Lab. of Parallel and Distributed Computing
Lab. of Software Engineering for Complex Systems
College of Computer, National University of Defence Technology
ChangSha, 410073, China
hmwang@nudt.edu.cn

ABSTRACT

The continuous contribution from peripheral participants is crucial for the success of open source projects. Thus, how to identify the potential Long-Term Contributors (LTC) early and retain them is of great importance. We propose a prediction model to measure the chance for an individual to become a LTC contributor through his capacity, willingness, and the opportunity to contribute at the time of joining. Using data of Rails hosted on GitHub, we find that the probability for a new joiner to become a LTC is associated with his willingness and environment. Specifically, future LTCs tend to be more active and show more community-oriented attitude than other joiners during their first month. This implies that the interaction between individual's attitude and project's climate are associated with the odds that an individual would become a valuable contributor or disengage from the project. We evaluated our prediction model by using the 10 cross-validation method. Results show that our model archives the mean AUC as 0.807, which is valuable for OSS projects to identify potential long-term contributors and adopt better strategies to retain them for continuous contribution.

Internetware '18, September 16, 2018, Beijing, China

ACM ISBN 978-1-4503-6590-1/18/09.
https://doi.org/10.1145/3275219.3275223

CCS CONCEPTS

• **Software and its engineering** → **Software development techniques**; *Collaboration in software development*;

KEYWORDS

Long Term Contributor, Developer behavior, GitHub, Open Source

ACM Reference Format:
Tao Wang, Yang Zhang, Gang Yin, Yue Yu, and Huaimin Wang. 2018. Who Will Become a Long-Term Contributor? A Prediction Model based on the Early Phase Behaviors. In *The Tenth Asia-Pacific Symposium on Internetware (Internetware '18), September 16, 2018, Beijing, China*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3275219.3275223

1 INTRODUCTION

For open source projects,it is crucial for them to attract and retain new contributors (newcomers) from the community continuously. Prior research found that every new contributor's participation can increase the chance of success for an open source project at 1.24 times [15]. However, it often takes a long time and much additional assistance for newcomers to become productive and undertake central tasks. 80% FLOSS projects failed not due to the quality of their products, but insufficient long-term participants [1].

Long-term contributors (LTCs), as a crucial factor for project's success, their formation is influenced by different dimensions, *i.e.*, capacity (*e.g.*, ability), willingness (*e.g.*,

attitude), and opportunity (*e.g.*, environment) [19]. And in open source communities, quite a large proportion of newcomers will not contribute to the project any more after their first participation [3]. Previous works have explored how newcomers join projects [18], overcome barriers [16], and make contributions [7]. However, few emperical works have been conducted on GitHub[1] for LTC prediction. As the biggest social coding community [9], GitHub has attracted millions of developers and open source projects, while only a small proportion of OSS in it achieve success. In this paper we aim to *understand what affects the chances for a new contributor to become a LTC on* GitHub *projects and how to identify the potential LTC at the early stage(the first few months after his first engagement)*. This study opens new opportunities for project leaders to recognize and retain the potential LTCs.

We firstly perform a preliminary study on a large and popular project on GitHub, Ruby on Rails[2] (Rails). We extract various dimensions of data to comprehensively profile the newcomers, including willingness and capacity, work burden and social status, environment and other feedback. Then we built a *Long-term contributor prediction model* to quantify the relationships between these dimensions and the probability that a newcomer would become a LTC. The results show the probability is highly associated with the contributor's attitude and environment. Specifically, at the time of joining, future LTCs tend to take more active role and show more community-oriented attitude than other newcomers. They also receive more attention from the community. Further, we used the model to predict the LTCs and the experimental result shows that it achieves the mean AUC as 0.807. The highlights of our contributions are:

- A set of quantitative metrics are designed to measure the developer's diversified behavior in the first-phase after joining in a new project;
- The behavior difference between LTCs and short-term contributors(STCs) in Rails are analyzed based on the designed metrics. We found that LTCs are much more active and diversified than STCs in the first-session time period(equal to 30 days);
- A long-term contributor prediction model is proposed to identify the potential LTCs, which achieves high performance;

This long-term contributor study on GitHub can assist project leaders to identify potential LTCs and adopt better strategies to retain them in the code contribution. With further investigation, it may eventually lead to long-term contributor identification tools, tailored to the GitHub community.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 describes our study methods, followed by our case study results (Section 4). Section 5 conclude the paper.

[1]https://github.com/
[2]https://github.com/rails/rails

2 RELATED WORKS

For open source projects, the crowds' participation and continuous contribution are of great importance. In this section, we review the previous works on the reason why volunteers would join in OSS, what will affect the newcomer's enthusiasm and how will they grow in these projects.

There have been rich studies on the motivation of developers to join in OSS communities. Alexander *et al.* [5] categorized the motivations into internal factors and external rewards. The typical internal factors include intrinsic motivation and altruism, and the external rewards can be expected future returns and personal needs. Hertel [6], Roberts [14] and Mair [10] *et al.* took the Linux, Apache and R communities respectively to study the volunteers' motivations and participation in these OSS.

Paulini *et al.* [13] conducted a research on what motivating continuous participation in online innovation communities. Their survey reveals that intrinsic motivations rated higher than extrinsic, and the passionate participants are either new members less than 1 month or long standing ones longer than 6 months. Ducheneaut *et al.* [3] found that 54% of newly registered developers never returned to the community after their first post in the Perl Project. Steinmacher *et al.* [17] found that less than 20% of newcomers became long-term contributors. This suggests that it is crucial for communities to take measures to retain new comers at the early stage.

To attract and retain newcomers, some researchers started to investigate what affects the probability that newcomers become long-term contributors. Panciera *et al.* [12] found that user's activity patterns, even in the earliest days, had an ability to predict future amount, quality and frequency of activity. Zhou and Mockus [20] proposed the findings suggest the importance of initial behaviors and experiences of new participants. They outlined empirically-based approaches to help the communities with the recruitment of contributors for long-term participation and to help the participants contribute more effectively. Pal *et al.* [11] found that by analyzing the changes in their behavior patterns in the first few weeks, we can distinguish experts from one another.

3 APPROACH

With the quick development of open source movement, the social coding community GitHub is formed, huge amounts of volunteers are attracted and millions of open source projects are developed in it. How to identify the potential LTCs and take measures to retain them at the early stage is quite important for the success of OSS projects and the GitHub community as well.

In this section, we firstly analyze the diversified behaviors of OSS developers in GitHub, and propose a model measure the various factors which may affect their choice for future contribution. Based on this model we design a prediction method using Regression Analysis to predict the potential LTCs.

3.1 The Diversified Behaviors of Developers

As an open source community which is distinguished from other OSS community by its social coding characteristics, GitHub provides *watch*, *star*, *fork*, *pull-request* and other mechanisms for crowds to contribute. For volunteers, they can contribute by proposing issues, posting comments, editing documents, submitting codes and so on.

Based on the events information recorded on GitHub API[3], we collect and present the most nine common and typical behaviors as follows:

- ***Forking (F)***: Developer cloned the entire Git repository of the project. After forking, the developer owns a copy of the original repository.
- ***Watching (W)***: Developer subscribed every event that happens to this project, including branch merging, version evolving and bug reporting.
- ***Subscribing (S)***: Developer subscribed issue to receive notifications that happens to the issue.
- ***Issue Comment submitting (IC)***: Developer submitted comment on the issues or pull requests, which do not reference a portion of the unified diff.
- ***PR review Comment submitting (PRC)***: Developer submitted comment on a portion of the unified diff in a pull request.
- ***Commit Comment submitting (CC)***: Developer submitted comment directly to a commit, which is outside of the pull request view.
- ***Issue submitting (I)***: Developer proposed a new issue like a new bug or a new requirement request to want other developers to solve.
- ***Pull Request submitting (PR)***: Developer sent a pull request to a project which includes the code commits.
- ***Commit submitting***: Developer sent a code commit directly to the main branch.

3.2 Study Dimensions

For a newcomer, both the internal and external factors can affect the probability for them to become LTCs. For each developer, we sort all his/her behavior records by the creation time. Then we divide each developer's behavior timeline from first to last behavior into N equal time sessions (each time session equals 30 days). We conduct a quantitative study based on four dimensions: behavior diversity, willingness and capacity, work burden and social status, environment and feedback.

3.2.1 Behavior Diversity. The behavior diversity of a given developer represents that how many features or mechanisms he/she experienced and how many types of tasks he contribute in the OSS project. For a developer, the more diversity he/she act , the more intensive he/she get to know the project, and the more probability he/she may contribute continuous.

[3]https://developer.github.com/v3/activity/events/types/

Behavior distance matrix

	C	PR	I	IC	PRC	CC	F	W	S
C	0	2	4	5	5	5	6	6	6
PR	2	0	4	5	5	5	6	6	6
I	4	4	0	5	5	5	6	6	6
IC	5	5	5	0	2	2	5	5	5
PRC	5	5	5	2	0	2	5	5	5
CC	5	5	5	2	2	0	5	5	5
F	6	6	6	5	5	5	0	4	4
W	6	6	6	5	5	5	4	0	2
S	6	6	6	5	5	5	4	2	0

Figure 2: The behavior distance matrix for calculating developer's behavior diversity. C: Commit; PR: Pull request; I: Issue; IC: Issue comment; PRC: PR review comment; CC: Commit comment; F: Fork; W: Watch; S: Subscribed.

Based on the method proposed by Karumur *et al.* [8], we build the behavior diversity metric in a manner that is tied to a hierarchical taxonomy of behaviors, *i.e.*, a taxonomy that is built specifically to group similar behaviors together and to separate dissimilar behaviors. To form this taxonomy, three of the authors firstly clustered the diversified behavior separately into as many clusters as would make sense to them. Then they discussed together on their clusters to reach an agreement. In the end, we get a hierarchy that depicted that relationship between various behavior types on GitHub as shown in Figure 1. In this hierarchy, all distinct behavior types appear as leaf nodes. Each internal node represents a hypothetical behavior type that encompasses all behavior types of its child nodes.

Based on the behavior hierarchy, we build a behavior distance matrix D to quantify the amount of dissimilarity between any two leaf nodes. In this matrix, the value of dissimilarity between two leaf nodes is simply the number of edges in the shortest path connecting them. Let D_{ij} denote the dissimilarity between leaf nodes i and j, which also represents the ij-th element of the matrix D.

We measure the behavior diversity a developer d as the normalized mean value of pair-wise dissimilarity between all behaviors in the set. If n is the number of distinct behavior types, then the behavior diversity value (***BDScore***) of the developer d is calculated as:

$$BDScore = \frac{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} D_{ij}}{n-1} \quad (1)$$

The behavior diversity is zero if a developer performs behaviors of only one type. behavior diversity increases as ancestral connection increases. In Figure 2, the set $\{C, F\}$

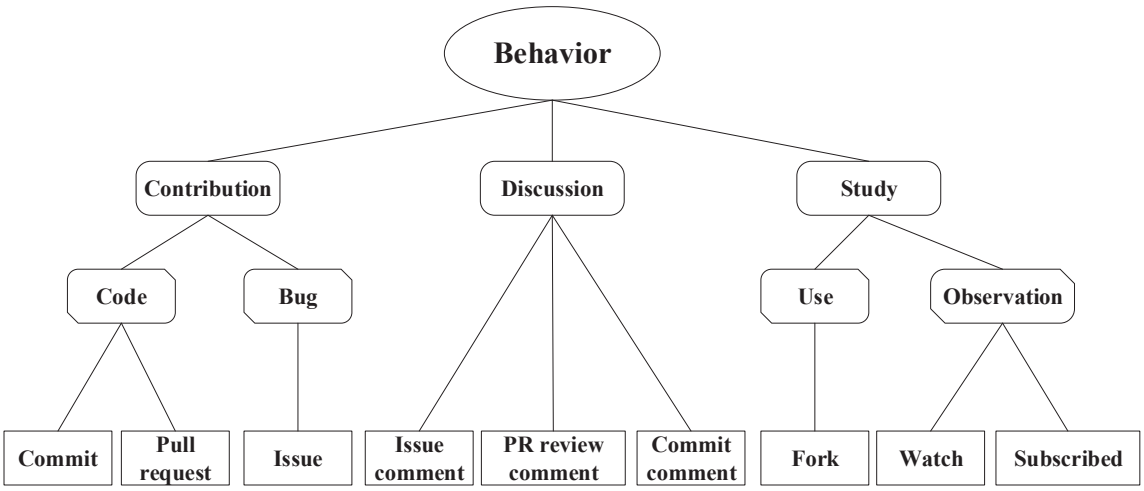

Figure 1: Classification tree of developer's behaviors.

is more diverse than the set $\{C, I\}$ which in turn is more diverse than the set $\{C, PR\}$. In other words, diversity of two leaf nodes whose parents are same is less diversity than two leaf nodes whose parents are different. Note that we are not interested in proportional abundance of a given leaf node (or behavior type), because all we care about is whether the user got an opportunity to use the feature at least once. Using this, we are interesting in predicting user churn, so we formulated the definition such that $\{C, I\}$, $\{C, C, I\}$ and $\{C, C, I, I\}$ are of equal diversity. In this work, the theoretical maximum for behavior diversity for our hierarchy is 42.5.

3.2.2 Willingness and Capacity. The probability for a new developer to become a LTCs is influenced by the willingness and capacity [19]. Willing and capacity are important factors to decide contributors' activeness in future, and they may also affect newcomers' activities.

To measure a newcomer's willingness and capacity quantitatively, we focus on the following three types of behaviors:

- **Number of comments (*nComments*):** Total number of comments created by developer, as a proxy of willingness. The more active the developers in discussion, the more willingness they contribute to the project.
- **Contribute code first (*codeFirst*):** True if the developer contributed code first in his participation, as a proxy of the willingness. Compared with commenting, coding first often suggests that the developer has a strong willingness to contribute.
- **Has PR got merged (*gotMerged*):** True if the developer has at least one pull-request be merged in his early participation, as a proxy of the capacity.

3.2.3 Workload and Social Status. We wonder whether work burden and social status are also associated with the likelihood of developers becoming long-term contributors. In this work, we use two features to measure the developer's workload and social status:

- **Number of owned projects (*nOwnProjs*):** Total number of projects that the developer have when he participates in the new one, as a proxy of the workload of the developer. The more projects a developer owned, the less time and effort he can spend on the new one.
- **Number of followers (*nFollowers*):** Total number of followers that developer have when he/she participated in a project, as a proxy of the social connections with other developers.

3.2.4 Environment and Feedback. Previous work showed that the project environment at the joining time have obvious impact on the contributors' behavior [19]. We explore whether project environment influences developers' behaviors:

- **Joining time (*joinTime*):** Time duration between the project creation and developer join in, in hours, as a control variable.
- **Project popularity (*popularity*):** Project's popularity when developer join in, as a proxy of project environment. Learning from Dabbish *et al.* [2], we define the project popularity as the sum of the number of stars, forks and watchers:

$$Popularity = \#Stars + \#Forks + \#Watchers \quad (2)$$

- **Got others' Comment (*gotComment*):** True if other developers commented in the developer's issue in the early stage of his participation, as a proxy of the other's feedback.

- **Got positive feedback (*positiveFeedback*):** To further measure the others' feedback, we used the sentiment analysis. In our study, we checked whether newcomer got the positive feedback in their early participation, we did the following text mining steps:
  1) *Step 1: Text extraction.*
  We extracted all the others' comment texts of each issue/pull request that the joiner submitted. And we combined those comment texts to a large textual string, as T;
  2) *Step 2: Tokenization.*
  Next, we divided the T into a set of tokens where a single token corresponds to a single term. This step also includes replacing all capitalized characters by their lower-cased ones;
  3) *Step 3: Stop-words removal.*
  Because terms like "the", "in" and "that" do not carry much specific information in the context of comments. Therefore, we removed all stop-words from the set of tokens based on a list of known stop-words;
  4) *Step 4: Stemming.*
  Due to each single term can be expressed in different forms but still carry the same specific information, we used the Porter stemmer algorithm to transform each term to its basic form;
  5) *Step 5: Sentiment analysis.*
  We used TextBlob tool [4] to get the sentiment score of the textual string. The sentiment score larger than 0 means positive feedback, otherwise means negative feedback.

3.3 Regression Modeling

To find out what factors affect the developers to become LTCs and to what extent these factors affect their choice, we propose a *Long-term contributor prediction model* based on the regression modeling method. We use the generalized linear regression modeling (package `glm` in R) with the "Logit" family of functions and log link function. The outcome (dependent) variables of the model is the tag of developer being a long-term contributor (***ltcTag***, 1 means True and 0 means False). Our independent variables come from different confound dimensions as described before.

During our modeling, we log-transform variables where needed to stabilize their variance and reduce heteroscedasticity. We remove the top 2% of the data to control outliers and improve model robustness. The variance inflation factors, which measure multicollinearity of the set of predictors in all our models, were safe, below 3. For each model variable, we report its coefficients, standard error, z-value, odds ratio, and significance level. We consider coefficients important if they were statistically significant ($p < 0.05$). In hypothesis testing, we used the non-parametric Wilcoxon-Mann-Whitney (WMW) test to test for a difference in the medians between two populations.

[4] https://textblob.readthedocs.io/en/dev/

4 EXPERIMENT SETTINGS

In this section we firstly present our research questions, describe the study case, and then give the evaluation metrics.

4.1 Research Questions

In this study, we aim to find out what factors may affect the newcomers' choice to be a LTCs, and investigate whether we can predict the LTCs by analyzing their early-phase behaviors. Particularly, we define the *long-term contributors* (LTCs) in a project as the developers who meet the following conditions:

- their active time in the project is more than one year;
- their contribution on source code was in the Top-30 among all the contributors in the project.

Note that the other developers who can not meet this standard will be viewed as *short-term contributors* (STCs).

To formulate our study, we focus on the following four correlated research questions:

- **RQ1:** *Newcomer churn.* How is the newcomer retention and churn?
- **RQ2:** *Behavior Difference between* LTCs *and* STCs. What are the behavior differences between the LTCs and STCs in the early-phase? Are these LTCs more active than STCs in the early-phase?
- **RQ3:** *Potential indicators for* LTCs. Which features of a newcomer in the early-phase could serve as good indicators of LTCs?
- **RQ4:** *LTC prediction model.* What if the performance of the LTC prediction model? Can it be useful for open source project leaders for identifying LTC at the early-phase?

4.2 Experiment Data

We choose the large and popular project on GitHub, Ruby on Rails (Rails), to perform our case study. Rails is a web-application framework that includes everything needed to create database-backed web applications with the Model-View-Controller (MVC) pattern. It is powering the GitHub infrastructure as well.

We first collected all historical data (before March 2016) of Rails project, including issues, comments, commits and issue events by using the GitHub API[5]. Then we extract all the stakeholders who have changed an artifact or interacted with another developer through the site.

Note that we ended our data collection on a certain date. Therefore, we do not have information about users whether or not they return to the system after one year. To avoid such bias, we removed the developers whose joining date was within one year before our deadline date. Totally, we collected 36,661 developers and their behavior data. Among them, 1,211 (3.3%) developers are long-term contributors. Table 1 shows the detailed statistics. We find that on average, the joinTime of each developer is 1,403 hours (median is

[5] https://developer.github.com/v3/

Table 1: Basic descriptive statistics result.

Feature	Mean	St.Dev	Min	Median	Max
#Developer sessions	6.2	15.34	0	0	96
#Comment	0.4	13.98	0	0	2,659
#Own projects	3.2	11.98	0	0	710
#Followers	9.7	110.63	0	2	16,361
BDScore	1.7	4.64	0	0	34.67
joinTime (hours)	1,403	689.90	0	1,462	2,530

Table 2: Confusion matrix used to calculate TPR and FPR.

		True Type	
		LTCs	STCs
Predicted Type	LTCs	*TP: True Positives*	*FP: False Positives*
	STCs	*FN: False Negatives*	*TN: True Negatives*

1,462 hours). On average, the BDScore of each developer is 1.7 (median is 0).

4.3 Evaluation Metrics

In our study, we used 10-fold cross validation methodology to train and evaluate our classifier. That is, we randomly partitioned discussions into 10 equal size sets. Then, we use nine of these sets as training data and one of them as test data. We repeated the cross-validation process 10 times, using each one of the sets exactly once as test data.

Table 2 represents all possible outcomes when making predictions of the developers. To evaluate the performance of our approach, we first used the two most commonly evaluation metrics: *Precision* and *Recall*.

- Precision: the percentage of newcomers predicted as either LTC which are correctly predicted. We define precision more formally as:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

- Recall: the percentage of all newcomers of LTCs that are actually predicted as LTCs. We define recall more formally as:

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

We also used an alternate technique, the Receiver Operating Characteristic (ROC), to evaluate the performance of our prediction model. The ROC compares the rate of true positives (TPR) with the rate of false positives (FPR) and is typically drawn as a curve [4]. Using this confusion matrix, the TPR and FPR can be calculated as:

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

The "area under the ROC curve" (AUC) is then a statistic summarizing the ROC curve in a single number representing the overall performance of the heuristic. This statistic represents the probability that the outcome of the heuristic is a better indication when compared to randomly choosing the severity. Random classification has an AUC value of 0.5 while the perfect heuristic has an AUC of 1, which means that the heuristic predicted the LTC correctly. Therefore, the higher AUC value is, the better the heuristic performs.

5 STUDY RESULTS

We conducted intensive experiments on the Rails project for the four research questions. The detailed results are presented in the subsections.

5.1 RQ1: Newcomer churn

In the first question, we want to study the developer churn in the Rails project. We firstly give a basic statistic analysis of the number of newcomers per month since the creation of Rails in GitHub.

Figure 3 shows that more than 250 newcomers would join in Rails every month. The average number is 400. This reveals that there are a lot of newcomers contributing to Rails which makes it success.

Then for each stakeholder, we extracted all his/her behavior data. We calculated the number of sessions that each developer have and counted the number of developers for the different number of sessions. We found that large number of developers drop out in their first few months. Particularly, the significant drop occurs right after the first month, as shown in the Figure 4.

Thus, we find that,

Table 3: Percentage developer churn after different sessions.

#Behavior Types in first session	1	2	3	4	5	6	7
#Developer	31,609	3,255	1,263	341	146	38	9
Developer Churn after 1 session	80.7%	67.0%	53.8%	47.5%	31.5%	28.9%	11.1%
Developer Churn after 3 sessions	81.9%	70.5%	59.8%	54.3%	42.5%	44.7%	33.3%
Developer Churn after 5 sessions	82.7%	73.6%	63.6%	59.2%	50.0%	50.0%	33.3%
Developer Churn after 10 sessions	84.6%	78.5%	71.0%	68.9%	60.3%	60.5%	55.6%
Developer Churn after 20 sessions	88.2%	86.0%	81.7%	82.7%	72.6%	73.7%	88.9%

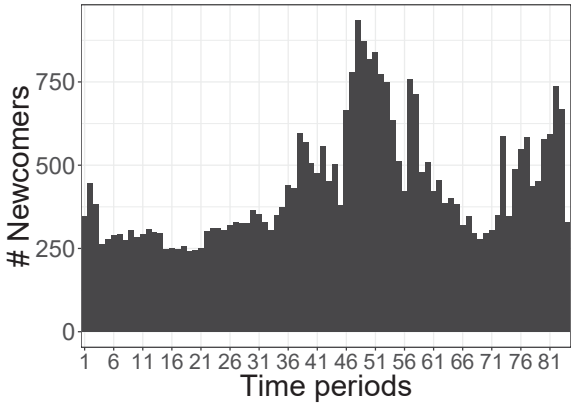

Figure 3: Number of new developers per month from Rails creation. Each time period equals one month.

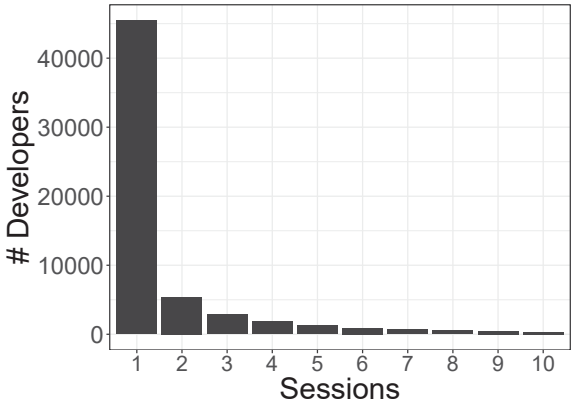

Figure 4: Number of developer who have been active for different lengths of time sessions, logged y-axis.

> ***Finding 1:*** *Although a large amount of developers joined in Rails, there is still a severe developer churn existing.*

Further, we investigate how developers behaviors affect the developer churn. We define percentage developer churn after the n months to be the number of developers who dropped out of the community after the n sessions over the total number of users who tried k behavior types in the first session where k = 1,2,...,7 (although participation in all 9 behavior types is theoretically possible, the developers in our dataset have participated in at most 7 behavior types by the end of the first session) and n = 1,3,5,10,20. The results are shown in Table 3. We find that, from the first session to the 20 sessions, the percentage of developer churn is increasing.

Also we find that 80.7% of developers that only performed one type of behavior churn after the first session. While in developers that performed two types of behaviors the percentage is decreased to 67.0%. In developers that had the full involvement behaviors (7 types), the percentage of developer churn is only 11.1%.

Thus, we find that,

> ***Finding 2:*** *The developer churn is worsening over time. Meanwhile, the lower diverse of developers behavior in the early phase, the greater the percentage of churn.*

5.2 RQ2: Behavior Difference between LTCs and STCs

In this subsection, our goal is to compare the behavior difference between the LTCs and the STCs, including the number of behavior activities of developers and the BDScore value of behavior.

(1) *The difference of number of behavior*

Figure 5 shows violin plots[6] of the distribution of the behavior quantity between LTCs and STCs. On average, each LTC contributes 2.2 behaviors, while each STC contributes 0.4 behaviors in their first-session. The WMW rank-sum test reveals that LTCs contributes more behaviors than STCs in the first-session (W=2.7×10^7; p<2.2×10^{-16}).

(2) *The difference of BDScore*

[6] https://en.wikipedia.org/wiki/Violin_plot

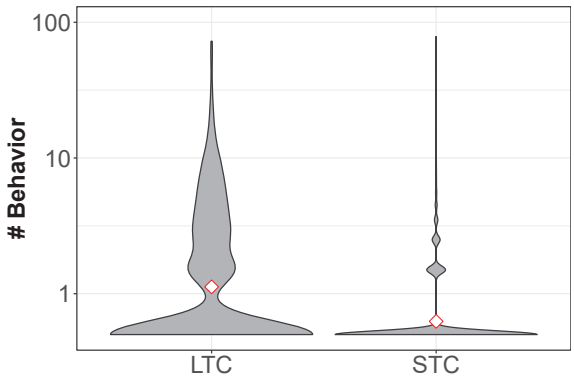

Figure 5: Number of behaviors of newcomers in the first-session.

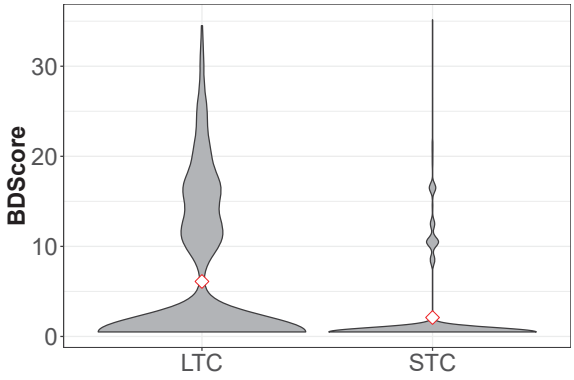

Figure 6: Behavior diversity score of newcomers in the first-session.

Figure 6 shows the distribution of the BDScore of LTCs and STCs. The average BDScore of LTCs is 5.6, while the value of STCs is 1.6. The WMW rank-sum test reveals that LTCs have more diverse behavior than STCs (W=2.8×10^7; p<2.2×10^{-16}).

Thus, we find that,

> ***Finding 3:*** LTCs *are more active and more diverse than* STCs.

Further, we compared the early feedback between LTCs and STCs. We find that 19.8% of LTCs got the positive feedback, while for STCs, only 4.3% of them got the positive feedback.

5.3 RQ3: Potential indicators for LTC

In this question, we seek to explore the potential indicators for LTCs by using regression modeling method. Table 4 shows the regression results of the *Long-term contributor model*. The fraction of total deviance explained by the model is 14.6%.

As expected, the factor ***joinTime*** has a significant negative effect on the outcome. The odds ratio for the control variable ***joinTime*** is $1-1.1^{-0.3864}\simeq 3.6\%$, holding all other variables constant, which means that the odds of initial changes are 3.6% lower for developers joining late to be a LTC. This verifies that joining time is an important environment control factor. Thus, *joining in a project earlier is good for the developer to grow to a LTC.*

The factor ***popularity*** has a significant negative effect on the *ltcTag*. The odds ratio for ***popularity*** is 1.9%, holding all other variables constant, which means that the odds of initial changes are 1.9% lower for developers joining late to be a LTC. Therefore, *the later joining in the project, the less likely to be a LTC.*

The factor ***nOwnProjs*** has a significant negative effect on the outcome. The odds ratio for ***nOwnProjs*** is 3.3%, holding all other variables constant, which means that the odds of initial changes are 3.3% lower for developers having many own projects to deal with to be a LTC. This indicates that *work burden would prevent newcomers from becoming an LTC.*

The factor ***nFollowers*** has a significant positive effect on the outcome. The odds ratio for ***nFollowers*** is $1.1^{0.7730}-1\simeq 7.6\%$, holding all other variables constant, which means that the odds of initial changes are 7.6% lower for developers having many social connections to be a LTC. Therefore, *the less burden and the more social connections, the more likely to be a LTC.*

The factor ***gotMerged*** has a significant positive effect on the outcome. The odds ratio for ***gotMerged*** is $e^{0.4877}-1\simeq 62.9\%$, holding all other variables constant, which means that the odds of initial changes are 62.9% higher for developers whose contribution being accepted to be a LTC. Thus, *the higher possibility that developer's contribution being merged, the more likely this developer being a LTC.*

The factor ***nComments*** has a significant positive effect on the outcome. The odds ratio for ***nComments*** is 1.6%, holding all other variables constant, which means that the odds of initial changes are 1.6% higher for developers discussing more to be a LTC. Therefore, *when a newcomer contributed many comments, he/she has a higher possibility to become a LTC.*

The factor ***codeFirst*** has a significant positive effect on the outcome. The odds ratio for ***codeFirst*** is 105.9%, holding all other variables constant, which means that the odds of initial changes are 105.9% higher for developers contributing codes first to be a LTC. Thus, *when contributing code first, the newcomer has a higher possibility to become a LTC in the future.*

The factor ***BDScore*** has a significant positive effect on the outcome. The odds ratio for ***BDScore*** is 2.2%, holding all other variables constant, which means that the odds of initial changes are 2.2% higher for developers behaving more diversified to be a LTC. Therefore, *the more active and diversity, the more likely to be a LTC.*

Table 4: Long-term contributor model. The response is *ltcTag*.

	Estimate	Std.Error	z value	Odds ratio
(Intercept)	-3.9540	0.0419	-94.268	
scale(log(joinTime+0.5))	-0.3864	0.0225	-17204	-0.0362***
scale(log(popularity+0.5))	-0.1985	0.0324	-6.128	-0.0187***
gotComment TRUE	0.5846	0.1170	4.996	0.7943***
positiveFeedback TRUE	0.3551	0.1178	3.013	0.4263***
scale(log(nOwnProjs+0.5)	-0.3554	0.0377	-9.418	-0.0333***
scale(log(nFollowers))	0.7730	0.0331	23.393	0.0764***
gotMerged TRUE	0.4877	0.1388	3.513	0.6286***
scale(log(nComments+0.5))	0.1644	0.0277	5.937	0.0158***
codeFirst TRUE	0.7224	0.1305	5.535	1.0594***
scale(log(BDScore+0.5))	0.2290	0.0355	6.460	0.0221***
Null deviance:				10,641.2
Residual deviance:				9,086.5
AIC:				9,108.5
Number of obj.:				36,661

Note: ***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

The factor ***gotComment*** has a significant positive effect on the outcome. The odds ratio for ***gotComment*** is 79.4%, holding all other variables constant, which means that the odds of initial changes are 79.4% higher for developers getting other's comment to be a LTC. Thus, *when a newcomer got other developers' comments, he/she is more likely to become a LTC.*

The factor ***positiveFeedback*** has a significant positive effect on the outcome. The odds ratio for ***positiveFeedback*** is 42.6%, holding all other variables constant, which means that the odds of initial changes are 42.6% higher for developers getting other's positive feedback to be a LTC. Therefore, *the more positive feedback, the more likely to be a LTC.*

Thus, we find that,

> ***Finding 4:*** *Developer behavior diversity, willingness and capacity, work burden and social status, environment and feedback, they all have effect on developer for being a LTC.*

5.4 RQ4: LTC prediction model

In this question, we aim to investigate the effectiveness of the Long-term contributor model when predicting real LTCs. Based on the Long-term contributor model, we conduct experiment on a proportion of developers to predict if they will become LTCs or not based on their early phase behaviors. The detailed results are shown in Table 5. We find that, through the 10-fold cross validation, the minimum value of precision is 0 at the 3 round. The maximum value of precision is 1 at the 4 round and the 9 round. For the value of recall, the minimum value is 0 at the 3 round, while the maximum value is 0.038 at the 6 round. For the AUC value, its minimum value is 0.757 at the 4 round, and its maximum value is 0.844 at the 8 round. Compared to the measurement of precision and recall, the fluctuation of the value of AUC seems to be smaller. Because most of developers in our dataset are STCs, using AUC can better reflect the effectiveness of our prediction model.

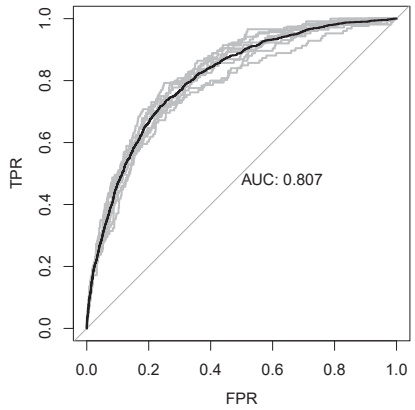

Figure 7: The ROC curve of our prediction evaluation.

Figure 7 shows the ROC curve of our 10-fold cross validation. The red curve shows the mean change. Overall, we see the performance of each validation is similar. The mean precision of our model is 0.500, the mean recall of model is 0.022, and the mean AUC of our model is 0.807, which indicates that our preliminary predictive model of LTCs could potentially be used in practice.

Thus, we conclude that,

> ***Conclusion:*** *Our prediction model has a robustness performance, which can be used to predict the LTC.*

Table 5: 10-fold cross validation result.

Round	1	2	3	4	5	6	7	8	9	10	Avg.
Precision	0.500	0.500	0.000	1.000	0.667	0.250	0.333	0.500	1.000	0.250	**0.500**
Recall	0.035	0.018	0.000	0.009	0.026	0.038	0.020	0.031	0.016	0.031	**0.022**
AUC	0.811	0.792	0.795	0.757	0.819	0.815	0.822	0.844	0.811	0.804	**0.807**

6 THREATS TO VALIDITY

Here, we discuss our threats associated with the methodology and analyses. In this work, we use precesion, recall, and ROC to evaluate the effectiveness of our model. These metrics are commonly-used in software engineering prediction tasks. Thus, the threat to our construct validity should be little.

We note that our model's fit to the data is around 15% of the deviance. That is not necessarily a problem for our purposes as we are only interested in the coefficients effect and not relying on the models to explain the full phenomena, which would require many more variables. Also in our prediction evaluation period, we proved that our model can perform a stable and good outcome.

Our external threat stems from the generalizability of our results. We evaluated our approach on a famous and large open-source projects on GitHub, *i.e.*, *Rails*. In future work, we plan to reduce this threat further by investigating the data from more projects.

7 CONCLUSION

We found that the main differences among participants were in their capacity, willingness and opportunity to contribute at the time of joining. It is also likely to help OSS communities to adopt better strategies to attract and retain newcomers. Specifically, the probability of staying longer is associated with how much value a new participant provides to the project by commenting, putting more effort into issue reports, and by the amount of attention the project provides to the newcomer. Ironically, it is during times when projects are popular, thus overwhelming the mentors, the community needs to put extra effort to retain newcomers.

8 ACKNOWLEDGMENT

This research is supported by Laboratory of Software Engineering for Complex Systems as well as National Grand R&D Plan (Grant No. 2018YFB1003903) and the National Natural Science Foundation of China (Grant No.61502512, 61432020, 61472430, 61303064).

REFERENCES

[1] Jorge Colazo and Yulin Fang. 2009. Impact of license choice on open source software development activity. *Journal of the American Society for Information Science and Technology* 60, 5 (2009), 997–1011.

[2] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 1277–1286.

[3] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368.

[4] David M Green and John A Swets. 1966. *Signal detection theory and psychophysics*. Robert E. Krieger,. 1478–1481 pages.

[5] Alexander Hars and Shaosong Ou. 2002. Working for Free? Motivations for Participating in Open-Source Projects. *International Journal of Electronic Commerce* 6, 3 (2002), 25–39.

[6] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy* 32, 7 (2003), 1159–1177.

[7] Chris Jensen and Walt Scacchi. 2007. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 364–374.

[8] Raghav Pavan Karumur, Tien T. Nguyen, and Joseph A. Konstan. 2016. Early Activity Diversity:Assessing Newcomer Retention from First-Session Activity. In *Acm Conference on Computer-supported Cooperative Work & Social Computing*. 595–608.

[9] Antonio Lima, Luca Rossi, and Mirco Musolesi. 2014. Coding Together at Scale: GitHub as a Collaborative Social Network.. In *Icwsm*.

[10] Patrick Mair, Eva Hofmann, Kathrin Gruber, Reinhold Hatzinger, Achim Zeileis, and Kurt Hornik. 2014. Motives for Participation in Open-Source Software Projects: A Survey among R Package Authors. *Research Report* (2014).

[11] Aditya Pal, Shuo Chang, and Joseph A Konstan. 2012. Evolution of experts in question answering communities.. In *ICWSM*.

[12] Katherine Panciera, Aaron Halfaker, and Loren Terveen. 2009. Wikipedians are born, not made: a study of power editors on Wikipedia. In *Proceedings of the ACM 2009 international conference on Supporting group work*. ACM, 51–60.

[13] Mercedes Paulini, Mary Lou Maher, and Paul Murty. 2014. Motivating participation in online innovation communities. *web based communities* 10, 1 (2014), 94–114.

[14] Jeffrey A Roberts, Ilhorn Hann, and Sandra A Slaughter. 2006. Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science* 52, 7 (2006), 984–999.

[15] Charles M Schweik, Robert C English, Meelis Kitsing, and Sandra Haire. 2008. Brooks' versus Linus' law: an empirical test of open source projects. In *Proceedings of the 2008 international conference on Digital government research*. Digital Government Society of North America, 423–424.

[16] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F Redmiles. 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology* 59 (2015), 67–85.

[17] Igor Steinmacher, Igor Wiese, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Why do newcomers abandon open source software projects?. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*. IEEE, 25–32.

[18] Minghui Zhou and Audris Mockus. 2011. Does the initial environment impact the future of developers?. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 271–280.

[19] Minghui Zhou and Audris Mockus. 2012. What make long term contributors: willingness and opportunity in OSS community. In *International Conference on Software Engineering*. 518–528.

[20] Minghui Zhou and Audris Mockus. 2015. Who will stay in the floss community? modeling participants initial behavior. *IEEE Transactions on Software Engineering* 41, 1 (2015), 82–99.