A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects

Lingfeng Bao, Xin Xia, David Lo, Gail C Murphy

Abstract—The continuous contributions made by long time contributors (*LTCs*) are a key factor enabling open source software (OSS) projects to be successful and survival. We study GITHUB as it has a large number of OSS projects and millions of contributors, which enables the study of the transition from newcomers to *LTCs*. In this paper, we investigate whether we can effectively predict newcomers in OSS projects to be *LTCs* based on their activity data that is collected from GITHUB. We collect GITHUB data from GHTorrent, a mirror of GITHUB data. We select the most popular 917 projects, which contain 75,046 contributors. We determine a developer as a *LTC* of a project if the time interval between his/her first and last commit in the project is larger than a certain time T. In our experiment, we use three different settings on the time interval: 1, 2, and 3 years. There are 9,238, 3,968, and 1,577 contributors who become *LTCs* of a project in three settings of time interval, respectively.

To build a prediction model, we extract many features from the activities of developers on GITHUB, which group into five dimensions: developer profile, repository profile, developer monthly activity, repository monthly activity, and collaboration network. We apply several classifiers including naive Bayes, SVM, decision tree, kNN and random forest. We find that random forest classifier achieves the best performance with AUCs of more than 0.75 in all three settings of time interval for *LTCs*. We also investigate the most important features that differentiate newcomers who become *LTCs* from newcomers who stay in the projects for a short time. We find that the number of followers is the most important feature in all three settings of the time interval studied. We also find that the programming language and the average number of commits contributed by other developers when a newcomer joins a project also belong to the top 10 most important features in all three settings of time interval for *LTCs*. Finally, we provide several implications for action based on our analysis results to help OSS projects retain newcomers.

Index Terms—Long Time Contributor, GitHub, Prediction Model

1 INTRODUCTION

Open source projects can have many contributors, but only a small proportion of contributors typically stay with the project for a long time. These long time contributors (*LTC*) often contribute a large proportion of code [12], [20], [32], [42], [48]. They are also usually experienced developers with extensive project experience who play important roles in the success of the project. Their contributions not only involve code writing, but also other tasks, such as solving bug reports, performing code review, understanding requirements from users, and helping and encouraging newcomers. An open source project is more likely to be successful if it can attract talented developers and retain them to be *LTCs*.

Unfortunately, most of contributors in an open source project leave the project and do not become *LTCs*. Foucault *et al.* [16] report that more than 80% of developers are either newcomers or leavers based on the history data of five OSS projects. There are many factors that can affect a newcomer to an open source project to become a *LTC* including the personality and expectation of the contributors, the working environment of the project, and the difficulty of tasks. These factors can be potentially inferred from the development activities of newcomers, which gives us a chance to predict which newcomers will potentially become *LTCs* based on developer activities.

By identifying potential *LTCs* early, project maintainers can take some actions to retain more contributors for long time, such as providing more attention to them or allocating more resources to them. Additionally, project maintainers can understand what factors are more important to retain contributors then improve the project. To predict long-time contributors, researchers have considered several approaches. For instance, Zhou and Mockus [76] extracted multiple factors based on bug report data from two OSS projects (i.e., Mozilla and Gnome). These factors cover three dimensions: extent of involvement (the first month activities of a newcomer from the date they join), macro-climate (i.e., the overall environment of the project) and micro-climate (i.e., the environment of individual contributor) of the project. They built a logistic regression model to predict whether a newcomer will become a *LTC*. In this model, two important features where how a newcomer started contribution to a project (i.e, by contributing to an existing issue report or by creating a new report) and whether an issue report submitted by the newcomer was worked on and completed by the project. Asri *et al.* [13] used five long-lived OSS projects to explore the temporal dynamics of GITHUB communities by time series analysis. They found that the

- *Lingfeng Bao is with the School of Computer & Computing Science, Zhejiang University City College, China. E-mail: baolf@zucc.edu.cn*
- *Xin Xia is with the Faculty of Information Technology, Monash University, Australia. E-mail: Xin.Xia@monash.edu*
- *David Lo is with the School of Information Systems, Singapore Management University, Singapore. E-mail: davidlo@smu.edu.sg*
- *Gail C Murphy is with the Department of Computer Science, University of British Columbia, Canada. E-mail: murphy@cs.ubc.ca*
- *Xin Xia is the corresponding author.*

number of submitted commits is the most important factor that affects whether a newcomer shifts to a core OSS team member. These previous studies consider a small number of projects and focus on a limited set of information (bug reports used by Zhou and Mockus, and commits used by Asri *et al.*).

In this paper, we take a new direction to predict long-term contributors by building a prediction model based on the first month of development activities of newcomers with a much larger dataset from GITHUB. At present, GITHUB holds ∼67 million OSS repositories involving 24 million developers[1]. GITHUB also tracks and provides access to several development activities, including code commits, issues (bug reports), pull requests, and discussions among developers. In this study, we select 917 projects from GHTorrent [21] – a mirror of Github data – based on developer rating (number of stars). Similar to the study of Zhou and Mockus [76], we build a model to predict whether a developer will become an *LTC* of a GITHUB project based on his/her first month development activities in the project. For each contributor of a project, we extract 63 features based on the GITHUB data which belong to five dimensions: developer profile, repository profile, developer monthly activity, repository monthly activity, and collaboration network. Based on these extracted features, we investigate four research questions:

RQ1: Can we effectively predict whether a developer will become a long time contributor of a project soon after developer submits his or her first commit to the project?

We apply several classic classifiers including naive Bayes, SVM, decision tree, kNN and random forest. We evaluate these classifiers on three different settings of time interval for *LTCs* and find that we can effectively predict whether a newcomer will become a *LTC* based on our extracted features. The random forest classifier has the best performance, achieving an AUC score of more than 0.75.

RQ2: How effective are the prediction models built on all features compared with prediction models built on only a subset of features?

We build a random forest model based on features in each dimension. In terms of AUC, the random forest built on all the features on average improves the random forests built on features from individual dimensions by a substantial margin.

RQ3: Which features are most important in identifying developers who become long time contributors?

We apply correlation and redundancy analysis to better model the integrated impact of features on newcomers being *LTCs*. We find that the number of a contributor's followers when he/she joins the project is the most important feature in all time interval settings. The user age of a developer, a project's programming language, the number of commits and watchers of a project before a newcomer join the project, the number of commits of a developer and a project in his/her first month also belong to the top-10 most important features in all three settings.

RQ4: How effective is our proposed approach in cross-project, cross-programming-language and cross-project-size prediction?

To perform cross-project prediction, we use a single project to build a model to predict contributors in other projects, and use all the other projects to build a model to predict contributors in the remaining one. We also use two similar setting to perform cross-programming-language and cross-project-size prediction. The results show that the models built on our proposed features achieve good performance on cross project (programming language or project size) prediction.

Paper Contributions:

- We build a prediction model based on a total of 63 features to determine whether a newcomer will become a *LTC* in a GITHUB project. We extract features based on a developer's first month activities in GITHUB, which belong to five dimensions. We conduct an experiment on a total of 75,046 developers from 917 projects. The results show that our approach can effectively predict whether a newcomer will become a *LTC* soon after he/she submits his/her first commit to the project.
- We investigate the most important characteristics that impact a newcomer being a *LTC*. We find that the number of a contributors followers when he/she joins the project is the most important feature in all time interval settings. We also find that the models built on our proposed features achieve good performance on cross-project (programming language or project size) prediction.

Paper Structure: The remainder of the paper is structured as follows. Section 2 describes the experimental setup including constructed dataset, extracted features, prediction models, and evaluation metrics. Section 3 presents the results of analysis for the three research questions. Section 5 discusses implications and threats to validity. Section 6 reviews related work. Section 7 concludes the paper and discusses future directions.

2 EXPERIMENT SETUP

2.1 Dataset

In our study, we use data from GHTorrent [21]; in particular, we analyze GHTorrent's MySQL database snapshot of 2017/09/01. Table 1 presents the entities considered in our study. To create a dataset for this study, we first select top 1,000 projects sorted by the number of their stars[2]. To select projects that are appropriate for our prediction task, we exclude projects for which:

1) the programming language of the project is empty in the GHTorrent database. We make this choice because these kinds of projects are usually not related to software development. For example, the project `tt free-programming-books`[3] is a collection of programming books and `gitignore`[4] is a collection of `.gitignore` templates.
2) GITHUB is not used as the issue tracker system. For such case, we cannot get issue data for the prediction task. For

1. https://octoverse.github.com/

2. developers can keep track of a Github repository by starring it.
3. https://github.com/EbookFoundation/free-programming-books
4. https://github.com/github/gitignore

TABLE 1: GHTorrent schema entities used in our study

Entity	Description
projects	Github repository
users	Github users
commits	List of all commits for each project
commit_comments	Comments associated with each commit
issues	Issues that have been recorded for each project
issue_comments	Discussion comments on an issue
issue_events	List of events on an issue, e.g., opened, assigned, closed
pull_requests	List of pull requests for each project
pull_request_comments	Discussion comments on a pull request
pull_request_history	List of events on a pull request, e.g., opened, merged
watchers	Users that have starred (was watched) a project
followers	Users that are following another user

example, the source code of `linux`[5] is hosted in GITHUB but it uses Bugzilla as issue tracker system.

3) the project is forked from another project, which can be determined by the field `fork_from` in the table `projects` of GHTorrent.
4) the project has been deleted from GITHUB, which can be determined by the field `deleted` in the table `projects` of GHTorrent.

Applying these criteria results in 917 projects in total in our dataset. Next, for each project, we consider developers who have submitted at least one commit as contributors. For some developers, the time of their first commits in a project can be earlier than the creation time of their GITHUB accounts. This might be because some projects are migrated from other hosting platforms (e.g., GitLab, Bitbucket). We exclude these developers from the projects as we cannot obtain all of their activities in the project.

We define a long time contributor (*LTC*) to be a contributor who stays with a project for more than a certain time T, i.e., the time interval between the first commit and the last commit of a contributor to the project is larger than T. In this study, we use three different settings of time interval for *LTCs*: 1, 2, and 3 years.

We follow a right censoring method [40] to discard the developers who joined within time T of the data gathering time, which is as follows: given a time interval T for the definition of *LTC*, if the time interval between the first commit of a developer in a project and the GHTorrent data dump time (i.e., 2017/09/01) is less than T, we exclude the developer since we cannot determine whether he/she becomes a *LTC* of the project or not. Table 2 presents the number of *LTC* and *non-LTC* for different time interval settings. When the time interval requirement is increased from 1 year to 3 years, the total number of contributors decreases from 75,046 to 26,698 as the time period between many contributors joining a project and the GHTorrent data dump time is less than 3 years. The constructed data set is unbalanced. The ratios of *LTC* and *non-LTC* are approximately 1:7, 1:11, and 1:16 when the time interval for *LTC* is 1, 2, and 3 years, respectively.

Table 4 presents project statistics including contributors, commits, issues, and pull requests in our constructed dataset. The projects used in our study have different sizes. For example, the project `homebrew`[6], which is a package manager in macOS, has the most contributors (close to 3500 developers) while the project `playground`[7], which is a deep learning visualization tool for tensorflow, only has two contributors. On average, there are 120.54±263.33 (mean±standard deviation) contributors in a project. The values of the other three project statistics (i.e., commits, issues, pull request) also vary very much. On average, the projects have 3,697±8,628, 1,573±2,628, and 1,124±2,842 commits, issues, and pull requests, respectively. Table 3 shows the number of projects developed using different primary programming languages. The 917 projects in our constructed dataset are developed using 37 different programming languages. We aggregate the languages used in a small number of projects as *Others* in Table 3. Javascript is used by the most projects in our dataset (358 projects). Other widely used languages are Java (75), Python (61), Ruby (47), and Go (45). This data analysis shows that projects in the dataset are diverse.

TABLE 2: The number of contributors in different time intervals for definition of *LTC*

Year	#LTC	#non-LTC	Total
1	9,238	65,808	75,046
2	3,968	45,384	49,352
3	1,577	25,121	26,698

TABLE 3: The number of projects on different programming languages.

Lang.	#Project	Lang.	#Project
JavaScript	358	HTML	30
Java	75	CSS	27
Python	61	PHP	24
Ruby	47	Shell	19
Go	45	CoffeeScript	13
Objective-C	42	VimL	12
C++	38	TypeScript	11
Swift	35	C#	10
C	30	Others	41

2.2 Studied Features

In this study, to investigate whether a developer will leave an open source project or be a long time contributor of the project, we consider data that is related to the developer and the project in GHTorrent and extract 63 features along five

5. https://github.com/torvalds/linux
6. https://github.com/Homebrew/legacy-homebrew
7. https://github.com/tensorflow/playground

TABLE 4: Project statistics in constructed dataset.

	Contributor	Commit	Issue	Pull Request
Total	110,538	3,390,447	1,442,358	1,030,559
Mean	120.54	3,697.33	1,572.91	1,123.84
Median	41	1,013	673	315
Minimum	2	24	1	7
Maximum	3,475	114,576	31,584	34,062
Std.	263.33	8,628.02	2,628.28	2,841.75

dimensions that might affect a newcomer become a *LTC*. We describe the meaning of each feature in Table 5.

Developer Profile Dimension. This dimension refers to features extracted from the overall information of a newcomer when he/she submits the first commit to a project, which is dependent on his/her historical activities in GITHUB. We use eight features to measure a new developer's profile – *user_age*, *user_own_repos*, *user_watch_repos*, *user_contribute_repos*, *user_history_commits*, *user_history_issues*, *user_history_pull_requests*, and *user_history_followers*.

The *user_age* feature quantifies the number of days between the registration date of the new developer and the date that he/she joins a project, which might be an indicator of his experience. The *user_own_repos* feature quantifies the number of project owned by the new developers, which might indicate his/her use of GITHUB. Additionally, a repository owned by the new developer might be forked from other repositories. It is likely that he/she wants to use or contribute to the forked repository. The *user_watch_repos* feature quantifies the number of repositories watched by the new developer, which might indicate he/she is interested in these watched projects. The new developer is more likely to contribute to a watched project than unwatched projects. The *user_contribute_repos* feature quantifies the number of repositories that the new developer has contributed to, which measures his/her experience on contributing to OSS projects in GITHUB. The *user_history_commits*, *user_history_issues* and *user_history_pull_requests* features measure the history activities in GITHUB. The *user_history_followers* feature might be an indicator of the programming and social ability of the new developer. These features indicate a developer's professional experience, activeness in GITHUB and willingness to contribute to OSS projects. An experienced GITHUB contributor might be more likely to stay with an OSS project for longer time than a junior GITHUB user. Schilling *et al.* reported that the level of development experience and conversational knowledge is strongly associated with developer retention [51]. Therefore, we believe that these features might affect a newcomer to be a *LTC* of an OSS project.

Repository Profile Dimension. This dimension refers to features extracted from the overall information of a project when a newcomer submits his/her first commit, which is dependent on the historical activities of all contributors in the project. This dimension is similar to the macro-climate of the project used in the study of Zhou and Mockuss [76]. Thus, we collect several kinds of project information. First, we get the main programming language used in the project. As some languages are complex and difficult to learn, while some (e.g., simple script languages) can be mastered fairly easily, the difficulty of programming language might be a barrier of being *LTCs* for newcomers [63]. Second, for each project, we count the number of commits before target developer first commit and their corresponding comments. The number of commits might indicate the workload and activeness of the project community, and the comments are the responses from other developers in the project, which might indicate the reactivity of the project community. Based on the commits, we also get the number of contributors, and calculate several metrics on the number of contributors commits including *max, min, mean, median, std (aka. standard deviation)* since we want to get a more accurate distribution of commits in the project history. For example, although the *mean* of contributors' commits indicates the average workload of the contributors in the project, but if some core developers contribute the majority of commits, the mean of commits might be still very high. So, we calculate these different metrics. These metrics provide insight into the development style of a project. For example, a higher standard deviation of the number of contributors commits means that a small proportion of developers contribute most of the source code, possibly indicating the difficulty for newcomers to become core developers of the project. Third, we count the number of issues and their corresponding comments and events[8]. We also count the number of *assigned* and *closed* issue events, which might show whether the project developers fix bugs efficiently. Similar to issues, we count the number of pull requests and their corresponding comments and events. We also count the *merged* and *closed* events of pull requests. Finally, we count the number of users who watch the project, which is a proxy for the popularity of the project. The study of Zhou and Mockus has shown that popular projects with more contributors have a higher barrier for newcomers to be *LTCs* [76].

Developer Monthly Activity Dimension. This dimension refers to features computed based on the first month activities of a newcomer since he/she joins the project. The initial effort that a newcomer spends in an OSS project is an important indicator of being an *LTC* [13], [76]. The activities of a developer in a project can be inferred from the commits, issues, and pull requests he/she submitted. Thus, we compute the number of commits (*month_user_commits*), the number of issues (*month_user_issues*), and the number of pull requests (*month_user_pull_requests*), which is similar to *the number of tasks* in the study of Zhou and Mockus [76]. These three features are direct indicators of activities of a newcomer in the first month. Zhou and Mockus used *the duration of time between the newcomers first action until somebody responds* to measure the amount of attention from the community. Meanwhile, we count the number of received comments for his/her submitted commits, issues, and pull requests (i.e., the feature *month_user_commit_comments*, *month_user_issue_comments*, *month_user_pull_request_comments*). These received comments represent the feedback from the community, which indicates how welcoming and inclusive the project community is [33]. Finally, we count the number of *closed* and *assigned* events for a newcomers submitted issues,

8. please refer to https://developer.github.com/v3/issues/events/

TABLE 5: Features Potentially Affecting Developers Being Long Time Contributors

Demension	Factor Name	Explanation
Developer Profile	user_age	Number of days between the registration date of the new developer and the date that he/she joins the repository
	user_own_repos	Number of repositories the new developer owns when he/she joins the repository
	user_watch_repos	Number of repositories the new developer watches when he/she joins the repository
	user_contribute_repos	Number of repositories in which the new developer has submitted at least one commit when he/she joins the repository
	user_history_commits	Number of commits the new developer has submitted when he/she joins the repository
	user_history_pull_requests	Number of pull requests the new developer has submitted when he/she joins the repository
	user_history_issues	Number of issues the new developer has submitted when he/she joins the repository
	user_history_followers	Number of users who follow the new developer
Repository Profile	language	Main programming language used by the repository
	before_repo_commits	Number of commits that the repository have when the new developer joins
	before_repo_commit_comments	Number of commit comments that the repository has when the new developer joins
	before_repo_contributors	Number of contributors that the repository has when the new developer joins
	before_repo_contributor_{S}	Statistics of commits of contributors in the repository, where S can be max, min, mean, median, and std
	before_repo_issues	Number of issues that the repository have when the new developer joins
	before_repo_issue_comments	Number of issue comments that the repository has when the new developer joins
	before_repo_issue_events	Number of issue events that the repository has when the new developer joins
	before_repo_issue_events_closed	Number of issue *closed* events that the repository has when the new developer joins
	before_repo_issue_events_assigned	Number of issue *assigned* events that the repository has when the new developer joins
	before_repo_pull_requests	Number of pull requests that the repository has when the new developer joins
	before_repo_pull_request_comments	Number of pull request comments that the repository has when the new developer joins
	before_repo_pull_request_history	Number of pull request events the repository has when the new developer joins
	before_repo_pull_request_history_merged	Number of *merged* pull request events the repository has when the new developer joins
	before_repo_pull_request_history_closed	Number of *closed* pull request events the repository has when the new developer joins
	before_repo_watchers	Number of watchers the repository has when the new developer joins
Developer Monthly Activity	month_user_commits	Number of commits that the new developer submits to the repository in the first month
	month_user_commit_comments	Number of comments received in the commits submitted by the new developer in the first month
	month_user_issues	Number of issues that the new developer submits to the repository in the first month
	month_user_issue_comments	Number of comments received in the issues submitted by the new developer in the first month
	month_user_issue_events	Number of events received in the issues that the new developer submits to the repository in the first month
	month_user_issue_events_closed	Number of closed events received in the issues submitted by the new developer in the first month
	month_user_issue_events_assigned	Number of assigned events received in the issues submitted by the new developer in the first month
	month_user_pull_requests	Number of pull request that the new developer submits to the repository in the first month
	month_user_pull_request_comments	Number of comments received in the pull request submitted by the new developer in the first month
	month_user_pull_request_history	Number of pull request events received in the pull request submitted by the new developer in the first month
	month_user_pull_request_history_merged	Number of merged pull request events received in the pull request submitted by the new developer in the first month
	month_user_pull_request_history_closed	Number of closed pull request events received in the pull request submitted by the new developer in the first month
Repository Monthly Activity	month_repo_commits	Number of commits submitted to the repository in the first month that the new developer joins
	month_repo_commit_comments	Number of comments received in the commits submitted to the repository in the first month
	month_repo_contributors	Number of contributors who submitted at least one commits to the repository in the first month
	month_repo_contributor_{S}	Statistics of commits of contributors in the first month, where S can be max, min, mean, median, and std
	month_repo_issues	Number of issues submitted to the repository in the first month
	month_repo_issue_comments	Number of comments received in the issues submitted to the repository in the first month
	month_repo_issue_events	Number of events received in the issues submitted to the repository in the first month
	month_repo_issue_events_closed	Number of closed events received in the issues submitted to the repository in the first month
	month_repo_issue_events_assigned	Number of assigned events received in the issues submitted to the repository in the first month
	month_repo_pull_requests	Number of pull requests submitted to the repository in the first month
	month_repo_pull_request_comments	Number of comments received in the pull requests submitted to the repository in the first month
	month_repo_pull_request_history	Number of events received in the pull requests submitted to the repository in the first month
	month_repo_pull_request_history_merged	Number of merged events received in the pull requests submitted to the repository in the first month
	month_repo_pull_request_history_closed	Number of closed events received in the pull requests submitted to the repository in the first month
Collaboration Network	degree_centrality	these metrics are used to quantify a newcomer's degree of activity in the collaboration structure of an OSS project
	closeness_centrality	
	betweenness_centrality	
	eigenvector_centrality	
	clustering_coefficient	

and the number of *merged* and *closed* events for his/her submitted pull requests. These features are also similar to the feature whether or not the first reported issue gets fixed used in the study of Zhou and Mockus [76]. From these features, we can know the first experience of a newcomer in the project. For example, if his/her first pull request is *merged* into the main branch of the project, he/she may feel happy and is willingness to make more contribution for the project.

Repository Monthly Activity Dimension. This dimension refers to features based on the first month activities of all contributors in an OSS project after a newcomer joins the project. The activities of all contributors might indicate the working environment in an OSS project. The working environment and other contributors in a project might have an important effect on a developers working experience [4]. Thus, similar to the features in repository profile dimension and developer monthly activity dimension, we first count the number of commits (*month_repo_commits*), issues (*month_repo_issues*), and pull requests (*month_repo_pull_requests*) submitted to the project in the first month that a newcomer joins. These features gives a first idea about the volume of the development effort contributed by all contributors in the first month. For these commits (issues and pull requests), we count the number of corresponding comments, which might be an indicator of interactions among the project contributors. We also count the number of *closed* and *assigned* events for the issues and the number of *merged* and *closed* events for the pull requests submitted to the project in the first month that a newcomer joins. These features might be indicators of responsiveness of the project community. Finally, we count the number of contributors (*month_repo_contributors*), which is similar to the feature *project sociality* (measured by the number of participants in the first month) in the study of Zhou and Mockus [76]. This feature might indicate the activeness of the project community. Additionally, we also compute five kinds of statistics on the number of commits of contributors: *max, min, mean, median, std*, which is similar to repository profile dimension. However, these statistics indicate the current state of the project in the first month that a newcomer joins, while the statistics in repository profile dimension indicate the history state of the project. Additionally, Zhou and Mockus also use the minimum number of issues in

network to measure *the performance of peers* [76].

Collaboration Network Dimension. This dimension refers to features based on collaboration activities between a newcomer and other contributors in an OSS project in the first month after he/she joins the project. Zhou and Mockus also constructed the collaboration network based on bug reports and calculated several features including *the size of a persons workflow peer group* and *the social clustering* (referred to be the amount of replication among the workflow networks of peers) [76]. The underlying assumption is that if a newcomer is in a more central position in the collaboration structure of the OSS project (i.e., he/she has more interactions with different contributors), it is more likely that he/she has good working experience. Furthermore, a central position in a collaboration network may indicate a better environment for newcomers to learn and become more effective, which can increase his or her work satisfaction and willingness to stay [11], [33], [57]. Given a month of activity data of a project, we build a collaboration network using the comments associated with commits, issues, and pull requests. Then, we extract features of the newcomers corresponding node in the network. In particular, given a project and a newcomer, we first collect all the commits, issues, and pull requests of the projects and their corresponding comments for the first month since the newcomer submit his/her first commit. Then, we build a graph based on creators of commits/issues/pull requests and the developers who provide comments on the corresponding commits/issues/pull requests. In the graph, each node denotes a developer, and for each discussion on commits/issues/pull requests, we create a directed edge from each participant to its creator. Subsequently, we compute five social network features that are proposed by Zanetti*et al.* [74] using the Python package `NetworkX`:

- *degree_centrality*: the number of links incident upon a node (i.e., the number of ties that a node has), which can be can be interpreted either in terms of the potential impact of a node on other nodes or as the amount of information available to a node. The value of *degree_centrality* is normalized by dividing the number of nodes in the graph.
- *closeness_centrality*: the inverse of sum of all distances of a node to all the other nodes [17]. It quantifies the degree to which a node is close to all other nodes in a network. A higher value of *closeness_centrality* of a new developers node indicates that he/she is closer to all other developers in the collaboration network.
- *clustering_centrality*: the ratio between the number of triangles connected to a node and the number of triples that are centered around it, where a triple centered around it is a set of two edges that are connected to the node [50]. This feature quantifies the degree to which nodes in the network tend to cluster together. A higher value of *clustering_centrality* of a new developer indicates that he/she has higher dense of collaboration activities with his/her neighbors in the network.
- *betweenness_centrality*: the total number of shortest paths between all possible pairs of nodes that pass through that node [8]. A higher value of *betweenness_centrality* of a new developers node indicates that he/she has more control on the collaboration network since more information will pass through his/her node.
- *eigenvector_centrality*: a network metric that assigns scores to nodes in a network based on the concept that connecting to high centrality nodes increases the nodes centrality [6]. The *eigenvector_centrality* of a node is recursively computed by the centrality of the nodes direct neighbors as shown in Formula 1, in which $DN(n_i)$ denotes the set of direct neighbors of the node n_i and λ is the largest eigenvalue of the adjacency matrix of the network:

$$Ev(n_i) = \frac{1}{\lambda} \sum_{n_j \in DN(n_i)} Ev(n_j) \quad (1)$$

2.3 Prediction Model

We study different classifiers that are widely used in software engineering research [29], [31], [34], [36], [45], including Naive Bayes, Support Vector Machine (SVM), Decision Tree, K-Nearest Neighbor (kNN), and Random Forest.

Naive Bayes: Naive Bayes classifiers [24] are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. The major advantage of naive Bayes classification is its short computational training time, since it assumes conditional independence.

SVM: Support Vector machine (SVM) [67] is developed from statistical learning theory, and it constructs a hyperplane or a set of hyperplanes in a high- or infinite-dimensional space, which are used for classification. SVM selects a small number of critical boundary instances as support vectors for each label (in our case, the labels are *not-retained* and *retained*), and builds a linear or non-linear discriminant function to form decision boundaries with the principle of maximizing the margins among training instances belonging to the different labels.

Decision Tree: C4.5 is one of the most popular decision tree algorithms [24]. A decision tree contains nodes and edges; each node in the decision tree represents a factor in the input factor space, while each branch in the decision tree represents a condition value for the corresponding node. A decision tree algorithm classifies data points by comparing their factor with various conditions captured in the nodes and branches of the tree.

K-Nearest Neighbor: K-Nearest Neighbor is an instance-based algorithm for supervised learning, which delays the induction or generalization process until classification is performed [24]. We use the Euclidean distance as the distance metric, and since the performance of kNN may be impacted by different values of k, we set k from 1 to 10, and report the best performance (in terms of F1-score) among the 10 values of k.

Random Forest: Random forest is a kind of combination approach, which is specifically designed for the decision tree classifier [9]. The general idea behind random forest is to combine multiple decision trees for prediction. Each decision tree is built based on the value of an independent set of random vectors. Random forest adopts the mode of the class labels output by individual trees.

2.4 Evaluation Metrics

We use AUC, namely Area Under the receiver operating characteristic (ROC) Curve, to evaluate the effectiveness of the proposed prediction models. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) across all thresholds. The value of AUC ranges from 0 to 1. The higher an AUC value, the better the performance of a classifier. Previous studies consider an AUC value of 0.7 as a promising performance score [36], [44]. For the random classifier, the values of AUC are equal to 0.5 regardless of class distribution [15]. AUC has been widely used in many software engineering research, e.g. [34], [36], [44], [49].

We choose AUC as the performance measure for several reasons:

- Unlike threshold-dependent measures (e.g., F1-measures) that often rely on a probability threshold (e.g., 0.5) for constructing a confusion matrix, the AUC is an independent threshold measure, which considers all possible thresholds. For threshold-dependent measures, it is difficult to determine an appropriate threshold in some cases. Therefore, we use AUC to avoid the threshold setting problem. Several researchers also suggest to use threshold-independent measures (e.g., AUC) instead of threshold-dependent measures (e.g. F1-measures). For example, Lessmann et al. recommended AUC as the primary accuracy indicator for comparative studies [36]; Tantithamthavorn and Hassan suggested to use AUC to avoid conflicting conclusions [58].
- Due to class imbalance being observed in our dataset, the AUC is a better choice, because it is more robust towards class distribution than other measures such as F1-measure. Using F1-measure to compare two prediction models in unbalanced dataset is unfair [44], [46], while AUC is insensitive to class distribution [36].
- The AUC has a clear statistical interpretation [36]. In our context, it evaluates the possibility that a classifier ranks a randomly chosen developer who becomes a *LTC* higher than a randomly chosen developer who does not become a *LTC*. Since our motivation is to identify *LTC*, the AUC is an appropriate measure to evaluate the performance of the proposed prediction models in our study.

3 Experiment Results

To investigate whether we can predict a newcomer in a open source community to be a long time contributor based on our proposed features, we investigate four research questions.

RQ1 *Can we effectively predict whether a developer will become a long time contributor of a project soon after developer submits his or her first commit to the project?*

Motivation: Attracting and retaining new developers is important to the continuous development of an open source project. We would like to effectively predict whether developers will be long time contributors of a open source project soon after they submit the first commit to the project when there is still time to proactively create an environment to raise the probability of creating a *LTC*. We use our proposed features based on the activities of developers and apply different prediction models to investigate whether it is feasible to build accurate models that help to predict whether developers will become *LTCs*.

Approach: We use the implementations of the prediction models described in Section 2.4 provided as part of the Weka tool [23].

In our analysis, we combine all developers of the 917 projects as a whole dataset and keep the time order of joining the projects amongst developers. These developers are first sorted in chronological order of the time on which a developer joins the project, and the list is then divided into 10 non-overlapping windows of equal sizes. We use the first n windows as the training data and the remaining $10 - n$ windows as the testing data (n from 1 to 9). Finally, we measure the evaluation performance by AUC. Since the classifiers used in this study have configurable parameters, which might have impact on their performance on predicting LTCs [19], [30], [38], [39], we follow the parameter optimization method of Tantithamthavorn et al. [59] to identify the optimal settings for the classifiers used in our study. In many defection prediction studies [35], [36], [59], researchers have investigated the effectiveness of many classifiers techniques with different parameter settings. Thus, we believe that the parameter settings used in these studies are enough for the classifiers techniques used in this study for LTC prediction. Table 6 presents the candidate parameter settings for these classification techniques. For SVM and decision tree, there are several candidate techniques. SVM can use two kinds of kernel, i.e., linear kernel and Radial basisi function kernel. Decision tree has three classification techniques including J48, LMT, and CART. We repeat the evaluation for each parameter setting, then report the highest averaged AUCs with the most optimal parameter settings.

In this research question, we use three time interval settings to decide *LTC*: 1, 2, and 3 years. Thus, we repeat the evaluation for each setting. We also choose a baseline model, i.e., random prediction, to compare our proposed prediction models with. For random prediction, it randomly predicts developers being long time contributors of a project. The AUC of the random prediction model is 0.5 since it is a random classifier with two possible outcomes.

Results: Table 7, 8 and 9 present the results of AUC of each prediction model with parameter optimization for time interval of 1, 2 and 3 years, respectively. In terms of AUC, random forest achieves the best performance because its AUCs on all settings are larger than those of the other prediction models. The average AUCs of random forest considering a time interval of 1, 2 and 3 years for *LTC* prediction are 0.768, 0.783, and 0.802, respectively. Except when the training data is the first window of developers and the time interval to decide *LTC* is 2 and 3 years (see Table 8 and 9), all AUCs of the random forest model are larger than 0.7, which indicates promising performance. On the other hand, other classifiers also achieve acceptable performance, i.e., their averaged AUCs are close or larger than 0.7.

Table 10 presents the parameter settings for the best performance of each prediction model. For KNN and random forest, the prediction models achieve the best performance in the same parameter setting for time interval of 1, 2, and 3 years. On the other hand, the best performance of the

TABLE 6: The classification techniques with different parameters.

Classifier	Classification Techniques	Parameter	Parameter Description	Studied classification techniques with their default (in bold) and candidate parameter values
Naive Bayes	Naive Bayes (NB)	Laplace Correction	[N] Laplace correction (0 indicates no correction).	NB = {0}
		Distribution Type	[L] TRUE indicates a kernel density estimation, while False indicates a normal density estimation.	NB = {TRUE, **FALSE**}
SVM	SVM with linear kernel (SVMLinear), SVM with Radial basis function kernel (SVMRadial)	Sigma	[N] The width of the Gaussian kernels .	SVMRadial={0.1, 0.3, **0.5**, 0.7, 0.9}
		Cost	[N] A penalty factor to be applied to the number of errors.	SVMRadial={0.25, 0.5, **1**, 2, 4} SVMLinear={**1**}
KNN	k-Neareast Neighbour (KNN)	#Cluster	[N] The number of non-overlapping clusters to produce.	KNN={**1**, 5, 9, 13, 17}
Decision Tree	C4.5-like trees (J48), Logistic Model Trees (LMT), Classification And Regression Trees (CART)	Complexity	[N] A penalty factor that is applied to the error rate of the terminal nodes of the tree.	CART={0.0001, 0.001, **0.01**, 0.1, 0.5}
		Confidence	[N] The confidence factor that are used for pruning (smaller values incur more pruning).	J48={**0.25**}
		#Iterations	[N] The number of iterations.	LMT={**1**, 21, 41, 61, 81}
Random Forest	Random Forest (RF)	#Trees	[N] The number of classification trees.	RF={**10**, 20, 30, 40, 50, 60, 70, 80, 90, 100}

[N] denotes a numeric value; [L] denotes a logical value.

TABLE 7: AUCs of each prediction model for *LTC* with 1 year of time interval.

#Training Window	Random	Naive Bayes	KNN	Decision Tree	SVM	Random Forest
1	0.500	0.668	0.631	0.691	0.657	0.735
2	0.500	0.686	0.637	0.697	0.710	0.745
3	0.500	0.684	0.663	0.680	0.713	0.753
4	0.500	0.726	0.668	0.735	0.736	0.767
5	0.500	0.705	0.678	0.735	0.711	0.774
6	0.500	0.704	0.678	0.726	0.728	0.770
7	0.500	0.715	0.678	0.729	0.731	0.777
8	0.500	0.718	0.687	0.751	0.742	0.791
9	0.500	0.705	0.682	0.756	0.719	0.802
Mean	0.500	0.701	0.667	0.722	0.716	0.768

TABLE 8: AUCs of each prediction model for *LTC* with 2 year of time interval.

#Training Window	Random	Naive Bayes	KNN	Decision Tree	SVM	Random Forest
1	0.500	0.635	0.658	0.549	0.650	0.694
2	0.500	0.662	0.673	0.702	0.714	0.744
3	0.500	0.696	0.689	0.708	0.719	0.779
4	0.500	0.668	0.739	0.755	0.749	0.794
5	0.500	0.695	0.738	0.781	0.738	0.809
6	0.500	0.716	0.757	0.795	0.746	0.824
7	0.500	0.740	0.780	0.823	0.776	0.839
8	0.500	0.750	0.778	0.803	0.802	0.857
9	0.500	0.725	0.766	0.833	0.836	0.878
Mean	0.500	0.699	0.731	0.750	0.748	0.802

TABLE 9: AUCs of each prediction model for *LTC* with 3 year of time interval.

#Training Window	Random	Naive Bayes	KNN	Decision Tree	SVM	Random Forest
1	0.500	0.581	0.630	0.643	0.647	0.684
2	0.500	0.658	0.668	0.702	0.684	0.712
3	0.500	0.677	0.690	0.705	0.729	0.751
4	0.500	0.681	0.669	0.726	0.712	0.743
5	0.500	0.715	0.678	0.775	0.718	0.775
6	0.500	0.721	0.738	0.777	0.784	0.806
7	0.500	0.712	0.754	0.799	0.780	0.842
8	0.500	0.732	0.762	0.745	0.795	0.845
9	0.500	0.756	0.799	0.813	0.824	0.884
Mean	0.500	0.692	0.710	0.743	0.742	0.783

other classifiers (i.e., Naive Bayes, decision tree, and SVM) is achieved in different parameter settings for different time intervals.

To investigate whether random forest has improvement in AUC compared to other prediction models, we apply Wilcoxon signed-rank test [65] with Bonferroni correction [2]. We also use Cliff's delta [10][9], which is a non-parametric effect size measure, to show the effect size of the difference between the results of two prediction models. Table 11 shows adjusted p-values and Cliff's delta for random forest compared with other prediction models. The improvements on AUC are all statistically significant at the confidence level of 99% and of a large effect size except for decision tree and SVM in time interval of 3 years (medium effect size). Therefore, we find that random forest has the

9. Cliff defines a delta which is less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474 and above 0.474 as negligible, small, medium, large effect size, respectively

TABLE 10: The parameter settings for the best performance of each prediction model.

	1 Year	2 Year	3 Year
Naive Bayes	Distribution Type=TRUE	Distribution Type=FALSE	Distribution Type=FALSE
KNN	#Cluster=17	#Cluster=17	#Cluster=17
Decision Tree	Classification Technique=LMT #Iterations=21	Classification Technique=LMT #Iterations=21	Classification Technique=CART Complexity=0.001
SVM	Classification Technique=SVMRadial Sigma=0.9 Cost=0.5	Classification Technique=SVMRadial Sigma=0.9 Cost=0.25	Classification Technique=SVMRadial Sigma=0.9 Cost=0.25
RF	#Trees=100	#Trees=100	#Trees=100

TABLE 11: Adjusted p-values and Cliff's delta for random forest compared with other prediction models on AUCs

	1 Year	2 Year	3 Year
Naive Bayes	1 (Large)**	0.83 (Large)**	0.73 (Large)**
KNN	1 (Large)**	0.73 (Large)**	0.60 (Large)**
Tree	0.85 (Large)**	0.83 (Large)**	0.33 (Medium)**
SVM	0.95 (Large)**	0.53 (Large)**	0.36 (Medium)**

**p<0.01

best performance and can effectively predict whether a developer will be a long time contributor of a project.

RQ2 *How effective are the prediction models built on all features compared with prediction models built on only a subset of the features?*

Motivation: In our study, we extract 63 features from five dimensions, i.e., developer profile, repository profile, developer monthly activities, repository monthly activities, and social network. In this research question, we investigate whether the prediction model benefits by using all features – as compared with its subsets.

Approach: As we find that random forest achieves the best performance in RQ1, we build random forest models (the parameter #*Tree* is set to 100) based on subsets of dimension features. For each dimension, we have two experiment settings: one is that building a random forest model only based on the features in it, and the other one is that building a random forest model based on the features in the other four dimensions. We use the same setup as RQ1 to evaluate the performance of prediction models and the reported AUCs are the average of nine runs of validation. Then we compare the AUC of the random forest model built on all features with those built on a subset of our proposed features. We also apply Wilcoxon signed-rank test with Bonferroni correction to investigate whether the improvements of the random forest built on all features over the five models are statistically significant. And we use Cliff's delta to measure effect size. We repeat this process for each setting of time interval setting for *LTC* prediction.

Results: Table 12 presents the results of average AUCs of the random forest models built from the subsets of features for all three settings of time interval for *LTCs*, and Tablee 13 present the corresponding adjusted p-values and Cliff's delta for the random forests built on all features compared with the random forests built on a subset of features. Comparing with the prediction models built from one dimension of features, the random forest models built on all the features achieve much higher values of AUCs (i.e., >0.75) for all three settings. Considering all feature subsets, the deltas in AUC are large. The largest delta (i.e., 36.9%) is for features in the collaboration network dimension considering the time interval of 1 year for *LTC* prediction, while the smallest delta (i.e., 6.1%) is for features in the developer profile dimension considering the time interval of 3 years for *LTC* prediction. Comparing with the prediction models built from four dimension of features, the random forest models built on all the features also achieves better performance except that built from the features without repository profile dimension features considering the time interval of 3 years for *LTC* prediction. But the delta is very small (i.e., 0.2%) and the difference is not significant and the effect size level is negligible. Thus, combining the five dimensions of features is beneficial in improving the effectiveness of the random forest model.

TABLE 12: AUCs of the random forest models built using subsets of dimension features

	1 Year	2 Year	3 Year
All	0.768	0.802	0.783
Dev. Profile	0.669	0.706	0.722
Repo. Profile	0.631	0.630	0.577
Dev. Mon. Act.	0.635	0.612	0.603
Repo. Mon. Act.	0.662	0.674	0.659
Collab. Network	0.372	0.571	0.525
˜Dev. Profile	0.754	0.741	0.713
˜Repo. Profile	0.765	0.793	0.785
˜Dev. Mon. Act.	0.761	0.797	0.774
˜Repo. Mon. Act.	0.757	0.771	0.746
˜Collab. Network	0.744	0.759	0.719

TABLE 13: Adjusted p-values and Cliff's delta for random forest with all features compared with those built on a subset of features.

	1 year	2 year	3 year
Dev. Profile	1 (Large)**	0.83 (Large)**	0.58 (Large)**
Repo. Profile	1 (Large)**	0.98 (Large)**	0.95 (Large)**
Dev. Mon. Act.	1 (Large)**	1 (Large)**	1 (Large)**
Repo. Mon. Act.	1 (Large)**	0.93 (Large)**	0.90 (Large)**
Collab. Network	1 (Large)**	0.98 (Large)**	0.85 (Large)**
˜Dev. Profile	0.41 (Medium)**	0.53 (Large)**	0.46 (Medium)**
˜Repo. Profile	0.16 (Small)**	0.11 (Negligible)**	0.06 (Negligible)
˜Dev. Mon. Act.	0.21 (Small)*	0.06 (Negligible)*	0.14 (Negligible)*
˜Repo. Mon. Act.	0.28 (Small)**	0.31 (Small)**	0.26 (Small)**
˜Collab. Network	0.48 (Large)**	0.41 (Medium)**	0.48 (Large)**

*p<0.05 **p<0.01

RQ3 *Which features are most important in identifying developers who become long time contributors?*

Motivation: Although there are multiple factors that may affect a developer to be a long time contributor of a project, some factors might be more influential than others. In this research question, we investigate these factors. Open source communities can use the influential factors to adjust policies for developers to be long time contributors.

Approach: Random forest has been shown to have the best performance in predicting whether a newcomer will become a *LTC* in RQ1. Therefore, we only investigate the most important features considering the random forest model.

Step 1: Correlation Analysis. To find the correlations among the features, we construct a hierarchical overview of correlations among the features by taking advantage of a variable clustering analysis, which is implemented in a R package named **Hmisc**. In the hierarchical overview, the correlated features are grouped into sub-hierarchies. We exclude the correlated features following the procedure performed in previous studies [4], [60] – we randomly select one feature and remove the other features from a sub-hierarchy in which the correlations of features are larger than 0.7. At the end of this step, for different time intervals for *LTC* prediction, i.e., 1, 2, and 3 years, there are 40, 40, and 38 features left, respectively.

Step 2: Redundancy Analysis. After reducing collinearity among the features by correlation analysis, we identify redundant factors that do not have a unique signal relative to other features. Redundant factors in an explanatory model will interfere with one another, distorting the modeled relationship between the factors and predictors. We use the **redun** function in the **rms** R package to perform redundancy analysis. But no redundancy is found in this step, thus we need not remove any features.

Step 3: Identification of Important Features. There are 40, 40, and 38 remaining features for three time intervals for *LTC* prediction after we remove the redundant features. Then, we build the random forest model using the **bigrf** R package. To evaluate the importance of features, we use 10-fold cross-validation to evaluate the effectiveness of the model, which is different from the evaluation approach used in RQ1. In the training process, we use the **varimp** function in **bigrf** package to compute the importance of a factor, which is based on out of the bag (OOB) estimates. OOB is an internal error estimate of a random forest classifier [66]. The underlying idea is to check whether the OOB estimates will be reduced significantly or not when features are randomly permuted one by one. To avoid randomness involved in the experiments, we repeat this step 100 times. Thus, we build and run the prediction models 1,000 times.

We get an important value for each feature in each run of validation. We apply the ScottKnott ESD test [1] using the importance values from all 1,000 runs to determine the features that are the most important for the whole dataset. This test takes as input a set of distributions (one for each variable) and identifies groups of variables that are statistically significantly different from one another.

Step 4: Effect of Important Features. To understand the impact of these features, we apply the Wilcoxon rank-sum test [65] with Bonferroni correction [2] to analyze the statistical significance of the difference between *LTC* and *not-LTC* developers. We use Cliff's delta to measure the effect size of the difference between the two groups.

TABLE 14: Top 10 most important features (1 Year)

Group	Feature	δ
1	user_history_followers★	**-0.286 (small)****
2	month_user_commits★	**-0.333 (medium)****
3	language★	—
4	before_repo_watchers★	**0.239 (small)****
	user_age★	0.054 (negligible)**
	before_repo_contributor_mean★	-0.019 (negligible)*
5	before_repo_contributor_std	0.046 (negligible)**
	before_repo_commits★	0.056 (negligible)**
	month_repo_commits★	-0.120 (negligible)**
6	month_repo_pull_requests	-0.017 (negligible)*

*p<0.01 **p<0.001

Results: Table 14, 15, and 16 present the top 10 most important features in three time interval settings for *LTC* prediction (1, 2, and 3 years), respectively. In these tables, different groups of features whose importance values are statistically significant different from other groups of features (i.e., $p - value < 0.05$). The last column δ shows the p-values and Cliff's deltas. The symbol ★in these tables indicates that a feature belongs to the top 10 important features in all three time interval settings for *LTC* prediction. Since *language* is a non numeric feature, we do not compute the p-value and Cliff's delta for it. If $p - value < 0.01$ and $|\delta| > 0.147$ (i.e., statistically significant difference with at least a small effect size), the text in the last column is in bold font. A positive value of Cliff's delta means that developers who do become *LTCs* of a project have significantly lower value on this feature, while a negative value of Cliff's delta means developers who become *LTCs* of a project have significantly higher value on this feature.

There are eight features that belong to the top 10 most important features in all three settings, i.e., *user_history_followers*, *user_age*, *language*, *before_repo_watchers*, *before_repo_contributor_mean*, *before_repo_commits*, *month_user_commits*, and *month_repo_commits*.

Among these features, *user_history_followers*, i.e., the number of followers, is the most important feature which affects the random forest model to differentiate *LTC* developer from *not-LTC* developers in all three settings. The statistical test shows that developers who become *LTCs* of a project have a significantly larger number of followers. The more followers a developer has, the more active and experienced he/she likely is. This finding indicates that developer activeness and experience have a big impact on developers' willingness for continually contributing to a project for a long period of time. The feature *user_age* is also an indicator of developers' experience, which have a potential impact on the models.

The feature *language*, i.e., programming language, might have potential impact on developers being *LTCs*. For example, Table 17 shows 10 programming languages with the largest number of contributors and the corresponding percentages of *LTCs* in 1 year time interval for *LTCs*. C/C++,

TABLE 15: Top 10 most important features (2 Year)

Group	Feature	δ
1	user_history_followers★	**-0.390 (medium)****
2	language★	—
3	before_repo_contributor_mean★	-0.071 (negligible)**
4	before_repo_commits★	0.064 (negligible)**
5	month_user_commits★	**-0.343 (medium)****
6	user_age★	-0.087 (negligible)
7	before_repo_watchers★	**0.194 (small)****
8	before_repo_pull_requests	**0.299 (small)****
9	month_repo_contributor_mean	**-0.179 (small)****
10	month_repo_commits★	-0.022 (negligible)

*p<0.01 **p<0.001

TABLE 16: Top 10 most important features (3 Year)

Group	Feature	δ
1	user_history_followers★	**-0.390 (medium)****
2	language★	—
3	before_repo_contributor_mean★	-0.071 (negligible)**
4	user_age★	0.087 (negligible)**
	before_repo_commits★	0.064 (negligible)**
5	month_user_commits★	**-0.343 (medium)****
6	month_repo_contributor_mean	**-0.179 (small)****
7	before_repo_pull_requests	**0.299 (small)****
8	month_repo_commits★	-0.022 (negligible)
	before_repo_watchers★	**0.194 (small)****

*p<0.01 **p<0.001

Java, and PHP have much higher percentages of *LTCs* than Javascript, HTML, and Shell. This might be because C/C++, Java, and PHP have been already applied in many industrial projects successfully, which require more complex technical and business knowledge. On the other hand, scripting language (e.g., Javascript, HTML, and Shell) is much easier to learn and the projects using these languages develop very fast so that the projects using these programming languages have more new contributors.

The feature *before_repo_contributor_mean*, *before_repo_commits*, and *before_repo_watchers* belong to the repository profile dimension, which indicate the size and workload of a project before a newcomer join the project. The feature *before_repo_watchers* is statistically significant and has positive values of Cliff's delta in all three settings, which indicates that it is difficult for newcomers to be *LTCs* in a mature and popular open source project, which usually have many watchers. The underlying reason might be that newcomers cannot get more resources and attention from project maintainers when they join a mature and large OSS project.

The feature *month_user_commits*, which belongs to developer monthly activity dimension, is statistically significant and has negative value of Cliff's delta in all three settings. The developers' work in the first month is an important indicator to predict whether they would become *LTCs*. Meanwhile, the feature *month_repo_commits* indicates the workload of a project in the first month that a newcomer joins.

We also find that the workload of other project members in the first month have a potential impact on newcomers being *LTCs*, i.e., *month_repo_contributor_mean* is statistically significant and has a negative value of Cliff's delta in Table 15 and 16.

Although there are no features of collaboration network dimension in the top 10 most important features for all the settings, we find that the differences between *LTCs* and *non-LTCs* on *eigenvector_centrality*, *betweenness_centrality*, and *clustering_coefficient* are statistically significant and Cliff's deltas are smaller than zero and in small level. This indicates that developers who have more interactions with others are more likely to be *LTCs*.

TABLE 17: The ratio of *LTCs* for different program language (1 Year).

Lang.	LTC (%)	#Contributor
JavaScript	9.29%	20,477
Ruby	12.35%	13,177
Python	12.77%	8,112
Go	12.75%	5,074
Java	14.55%	4,523
PHP	14.59%	4,454
C++	18.53%	4,090
HTML	6.53%	2,036
Shell	7.46%	1,716
C	14.48%	1,561

RQ4 *How effective is our proposed approach in cross-project, cross-programming-language and cross-project-size prediction?*

Motivation:

In the experiment setting of the previous research questions, we built prediction models based on all the historical data of the projects in our dataset. However, a newly built project may not possess enough historical data to build a model. Hence, in this research question, we want to know the effectiveness for predicting developers becoming long time contributors based on our proposed features in a cross-project setting. Additionally, the 917 projects used in this study can be divided into different groups in different dimensions such as programming language and project size. Since programming language evolves rapidly, there could be an emerging programming language in the future but not many projects on it yet, we want to investigate whether our approach is effective in cross programming language setting. Furthermore, the project size in GITHUB follows a long tail distribution (i.e., there are more than 96 millions repositories in GITHUB[10] but only a small fraction of them have more than 1,000 contributors). There are little data for larger projects, so we want to see if our approach is sensitive to project size.

Approach: For each project, we first build a prediction model based on the data from it then we use the model to predict the label of developers in other projects. Since there are too few developers in some projects of our dataset to build a prediction model, we only use the projects that have more than 500 developers in this setting of cross-project prediction. Consequently, we get 21 projects with different programming languages and numbers of contributors as shown in Table 18. The project *pandas* and *homebrew* has the minimum and maximum number of contributors (i.e., 660 and 3475 contributors), respectively (see the 3rd column in Table 18). For these 21 projects, we also build a prediction

10. https://octoverse.github.com/projects#repositories

TABLE 18: The results of cross project prediction. #Contr. = Number of contributors, AUC (Single) = mean and standard deviation of AUCs when using developers that belong to one project/language as training data to predict developers that belong to another one project/language. AUC (Combined) = AUCs when using developers that belong to all the other projects/languages as training data to predict developers that belong to the remaining one.

Project	Language	#Contr.	AUC (Single)	AUC (Combined)
pandas	Python	660	0.630±0.079	0.689
spacemacs	Emacs Lisp	725	0.658±0.074	0.720
spree	Ruby	749	0.652±0.102	0.783
rust	Rust	827	0.654±0.062	0.643
kubernetes	Go	853	0.660±0.081	0.672
react	JavaScript	880	0.638±0.063	0.579
oh-my-zsh	Shell	902	0.603±0.061	0.638
scikit-learn	Python	1,019	0.675±0.055	0.778
docker	Go	1,022	0.655±0.093	0.704
gitlabhq	Ruby	1,036	0.604±0.075	0.682
framework	PHP	1,131	0.648±0.070	0.706
odoo	Python	1,162	0.648±0.084	0.714
salt	Python	1,206	0.594±0.080	0.722
Spoon-Knife	HTML	1,278	0.538±0.085	0.737
symfony	PHP	1,350	0.652±0.060	0.726
angular.js	JavaScript	1,796	0.686±0.059	0.615
react-native	Java	1,966	0.609±0.090	0.696
ansible	Python	1,975	0.629±0.061	0.770
rails	Ruby	2,655	0.653±0.064	0.771
homebrew-cask	Ruby	3,099	0.588±0.066	0.640
homebrew	Ruby	3,475	0.667±0.077	0.786

TABLE 19: The results of cross language prediction.

Language	#Contr.	AUC (Single)	AUC (Combined)
Objective-C	1,117	0.679±0.057	0.860
Shell	1,758	0.650±0.101	0.715
HTML	2,280	0.683±0.095	0.703
Scala	2,293	0.717±0.059	0.658
PHP	4,654	0.702±0.063	0.737
C++	5,095	0.703±0.064	0.697
Go	5,209	0.693±0.078	0.787
C	5,541	0.695±0.068	0.677
Java	5,599	0.707±0.064	0.755
Python	8,544	0.705±0.074	0.741
Ruby	1,3656	0.702±0.070	0.701
JavaScript	2,1439	0.695±0.071	0.830

model based on the data from all the other projects in the whole data, then use the model to predict the label of developers in this project. We only use random forest to build prediction models as it has been proved to have best performance in RQ1.

For the cross-language setting for *LTC* prediction, given one programming language, we build a prediction model based on all the projects using it as the main programming language[11], and then we use the model to predict the label of developers in the projects using another programming language. Similarly, we also build a prediction model based on the projects using all the other languages, then use the model to predict the label of developers in the projects using the remaining one. In this setting, we select 12 programming languages that have more than 1,000 instances as shown in Table 19. We also list the number of contributors for each programming language (see the 2nd column in Table 19).

For the cross-project-size experiment for *LTC* prediction, we first sort all the projects by the number of contributors and divide them into four equal-sized groups using quantiles. The four group are denoted as Q1, Q2, Q3, Q4, respectively. Then, for each group, we build a prediction model based on the projects in it and use the model to predict the label of developers in the projects of other groups. Similarly, we also build a prediction model based on the projects of all the other groups, then use the model to predict the label of developers in the projects of the remaining one. We list the total number of contributors in the four groups in Table 20. As shown in the table, we find that the first group (Q1) has much less contributors than the last group (Q4).

Results: Tables 18 and 19 present the results of cross-project prediction and cross-programming language prediction, respectively. We refer to single cross-project (cross-language or cross-project-size) prediction setting as building a model based on a single project or a single dimension of projects to predict another single project or dimension of projects; on the other hand, we refer to combined cross-project (cross-language or cross-project-size) prediction setting as building a model based on all the other projects to predict the remaining one.

Cross project: In *single* cross project prediction setting, the mean of AUCs over all the projects in Table 18 is equal to 0.635, and the maximum and minimum AUCs are 0.686 and 0.538, respectively. The standard deviations of AUCs are not very large, which indicates that the performance of cross project prediction is stable. For some projects such as `scikit-learn` and `angular.js`, the prediction performance is promising, i.e., close to 0.70. In *combined* cross project prediction setting, our prediction model achieves a promising performance, i.e, 0.703 in average. The maximum and minimum AUCs are 0.786 and 0.615, respectively.

Cross-language setting: In *single* cross-language prediction setting, the mean of AUCs over all the languages in Table 19

11. The main programming language is determined by GitHub and we get the language of a project from the table `projects` in GHTorrent.

TABLE 20: The results of cross-project-size prediction.

Group	#Contr.	AUC (Single)	AUC (Combined)
Q1	1,557	0.761±0.063	0.897
Q2	4,597	0.792±0.090	0.864
Q3	11,408	0.821±0.072	0.835
Q4	57,484	0.861±0.020	0.719

is equal to 0.694. The maximum and minimum AUCs are 0.717 and 0.659, respectively. Meanwhile, the mean of AUCs in *combined* cross-language prediction setting is equal to 0.738, and the maximum and minimum AUCs are 0.860 and 0.658, respectively. All these values of AUCs are close to or more than 0.7, which indicates the prediction model achieves promising performance in cross-language prediction.

Cross-project-size setting: In *single* cross-project-size prediction setting, the maximum and minimum AUCs are 0.861 and 0.761, respectively. Meanwhile, the maximum and minimum AUCs in *combined* cross-project-size setting are 0.897 and 0.719, respectively. All these values of AUCs are larger than 0.7, which indicates the prediction model achieves promising performance in cross-project-size prediction.

4 Feedback from Developers

To investigate how developers participate in OSS projects and their opinions on the factors that affect developers being LTCs, we conducted a survey. The primary goal of the survey is to validate whether important features proposed in our approach are considered by OSS developers to have impact on whether developers become LTCs.

4.1 Survey Participants

We randomly sampled 1,250 individuals from 75,046 developers in our dataset. Among them, 250 recipients were chosen to be *LTCs* in time interval of 3 years, and 1,000 are from *non-LTCs* in time interval of 1 years. Out of these recipients, we received 26 responses from *LTCs* and 122 responses from *non-LTCs*, respectively, providing an overall response rate is 11.84%, which is similar to previous surveys performed on GITHUB [3], [5], [47]

4.2 Survey Design

Each survey recipient was sent an email in which the project name and recipients first comment in the project was mentioned. This email provided specific retrieval cues and context for the recipient (i.e., the project name and the recipient's first commit in the project) to help the recipient recall their previous work [61], [76]. Each recipient was asked about (see Table 20):

1) their role and working experience they started the project (Q1 and Q2).
2) their motivation for contributing to the project (Q3).
3) to rate their agreement on a 5-point scale the three most important features determined for RQ3 (i.e., *user_history_followers*, *language*, and *before_repo_contributor_mean*).
4) any other factors they believe can affect the chance that a newcomer will contribute for a long time.

4.3 Survey Results

As shown in Table 20, the majority of respondents (70.3%) were volunteers when they started to contribute we asked them about. Many of them were experienced developers: 34.6% and 47.5% of respondents had more than 6 years of work experience for LTCs and non-LTCs, respectively. Meanwhile, only a small fraction of respondents (12.8%) were junior developers. The top two factors cited that prompted respondents to start contributing to an OSS project are to report or fix a bug (39.2%) and to implement a new feature (26.4%).

For the three important features proposed by our approach, 58%, 64%, and 62% of respondents agree or strongly agree that they have big impact on a developer being a LTC. The overall scores are 3.56, 3.61, and 3.57, respectively. This indicates that the features proposed by our approach have the support of developers.

For the last open question, 42 out of 122 respondents gave an answer. We follow an open card sort approach [55] to identify three categories of factors that can affect the chance that a newcomer will contribute for a long time:

- Many survey respondents mention that **project quality** is an important factor. Project quality is related to the quality of code, documentation, build or setup process, etc. The comments that fall into this category include:

☞ *"An easy build process. Guides on how to contribute. Clean source code. Friendly community."*

☞ *"The processes employed by a project. If a project makes it difficult to get code merged - old projects are terrible for this often using mailing lists - then I think people lose their motivation to contribute. Conversely projects that make it easy to contribute, have good infrastructure and prompt code review help encourage repeat contributions."*

- We also find that characteristics of **project community** are influential. These correspond to community response to a newcomers first commit (issue or pull request), activity of project owner and pull request reviewers, mentors, etc. The comments that fall into this category include:

☞ *"I think the most important aspect is the community. How welcoming, how inclusive, how fun it is."*

☞ *"reactivity of the community , if your Pull Request wait too long you won't create a second one on this project"*

- Another category that we identify is **personal factors**, which is related to developers interest, time, needs, etc. The comments that fall into this category include:

☞ *"I stopped contributing to the kernel because I changed jobs, and my current job uses too old a kernel to push patches to upstream Linux. I contributed code to u-boot bootloader instead. Access to hardware and speed of kernel project are main barriers to continued contribution."*

☞ *"How important that project is to the contributor when I contribute to projects that arent particularly central to the kinds of work I do, I am less likely to see that as a project I want to continue to spend a lot of time with, or that I have a real connection with."*

Table 22 present the number of responses that belong to each category of factors that affect chances that a newcomer will contribute for a long time. Notice that one respondent could mention multiple factors in their answer, for example:

TABLE 21: The quesitons in the survey and the corresponding statistics.

	#LTC	#Non-LTC	All
#Response	26	122	148
Q1.What was your role for the project when you started to contribute the project (please refer to your project name in the email).			
An employee of a company that develops a project	2 (7.7%)	14 (13.3%)	16 (10.8%)
An employee of a company that works on / contributes to the project;	3 (11.5%)	13 (14.3%)	16 (10.8%)
A volunteer	18 (69.2%)	86 (87.8%)	104 (70.3%)
Others	3 (11.5%)	9 (9.2%)	12 (8.1%)
Q2. How many years of work experience in software development did you have when you started to contribute to the project?			
<1 year	3 (11.5%)	16 (13.1%)	19 (12.8%)
1 - 3 year	5 (19.2%)	27 (22.1%)	32 (21.6%)
4 - 6 year	9 (34.6%)	21 (17.2%)	30 (20.3%)
>6 year	9 (34.6%)	58 (47.5%)	67 (45.3%)
Q3.What prompted you to start contributing to an OSS project?			
It is a popular project, I wanted to contribute	4 (15.4%)	6 (4.9%)	10 (6.8%)
It has a good community with many interesting people	1 (3.8%)	1 (0.8%)	2 (1.4%)
I wanted to learn, and doing something in this project could achieve that	5 (19.2%)	13 (10.7%)	18 (12.2%)
I wanted to report or fix a bug in the project	8 (30.8%)	50 (41.0%)	58 (39.2%)
I wanted to implement a new feature that the project did not have	3 (11.5%)	36 (29.5%)	39 (26.4%)
My job was related to the project	4 (15.4%)	12 (9.8%)	16 (10.8%)
others	1 (3.8%)	4 (3.3%)	5 (3.4%)
Please indicate your agreement and disagreement		**Average Score**	
"A developer who is experienced and active in Github is more likely to be a long time contributor if that developer starts to contribute a new project"	3.15	3.65	3.56
"The programming language a project uses has an impact on a developer being a long time contributor"	3.23	3.69	3.61
"The development stage of a project has an impact on a developer being a long time contributor"	3.65	3.55	3.57
Open Question			
If you have more suggestions about which factors can affect the chances that a newcomer will contribute for a long time, please kindly advise.			

TABLE 22: The number of responses that belong to each category of factors that affect chances that a newcomer will contribute for a long time.

	LTC	non-LTC	All
ALl Response	10	32	42
Project Quality	4	14	18
Project Community	7	17	24
Personal Factor	4	11	15

☞ *"In my experience, I typically contribute to projects that I need for my work either development tools or components/libraries. Some factors that are important to me are the readability of the project and its code, the responsiveness of the members to issues, how well they organize and maintain the issues, responsiveness of members to pull requests (big one!), the existence of development documentation or tutorials, and simply how long the project is important to me if I stop using bootstrap in my work, I will stop contributing to it."*

From this feedback, we believe that the features proposed as part of our approach have modeled many factors deemed suitable by respondents. Comments related to project community characteristics are covered by features in repository profile dimension and repository monthly activity dimension. For example, the number of comments received in commits (issues or pull requests) is an indicator of how "warmly" a community welcomes a newcomer. The project quality factors are covered by features in the repository profile dimension. For example, a project with good quality usually has more watchers; one respondent said that *"I also believe that people are more likely to contribute to relevant projects. In other words, the most popular a project is, higher are the chances newcomers will help."* For personal factors, they are covered by features in the developer profile dimension.

4.4 Survey Implications

From the survey results, we find that several factors (project quality, project community, and personal factors) that may affect the chances that a newcomer will contribute for a long time have not been fully considered by our approach. These may result in incorrect predictions. Additionally, most of the factors considered by our approach are measured considering statistics obtained from counts of development activities (e.g., number of commits, etc.) happening in a project. Our approach did not consider detailed information, e.g., the contents of commit logs, the text contents in a pull request review, etc. Based on these, we list some additional steps that can potentially be considered (in a future work) to improve the accuracy of our approach:

- Project quality: To consider this factor for LTC prediction, there are a number of analyses that we can perform. For example, we can investigate the quality of the projects code base. The code base quality can be estimated in various ways, e.g., by counting the number of self-admitted technical debts [64], by running static analysis tools to identify code smell instances [62], etc. Additionally, we can evaluate the quality of the projects documentation by measuring the readability of the projects readme file or checking whether there is a wiki page for the project [28].

Moreover, we can also assess how thorough is the projects quality assurance process, e.g., by investigating the coverage of the test cases (if they exist) [25].
- Project community: We can extract additional information to better characterize and model the project community [56]. For example, we can use sentiment analysis to measure whether comments in the discussions are positive or not. By doing so, we may better characterize how welcoming the people in the project are and the culture of the project community.
- Personal factors: To model personal factors that affect a newcomer to be a LTC of a target project, we can analyze whether the other projects contributed by the newcomer are dependent on the target project. Thus, we can infer the likelihood of a newcomer to contribute to a project in a long term based on how important is the project to him/her.

Extracting the above features requires much additional effort that we consider to be beyond the scope of this work, and thus we leave their explorations for a future work.

5 Discussion

5.1 Comparison with *LTC* Prediction based on Bug Reports

The work based on bug reports conducted by Zhou and Mockus [76] is the most similar to our study. They also considered the activities in the first month that a newcomer joins and extracted nine features based on three dimensions: extent of involvement, macro-climate, micro-climate – see Table 23. Of the 9 features considered by Zhou and Mockus, 8 are similar to our features. For example, they considered the number of tasks measured by the number of comments while we measure this by the number of commits submitted by developers. They built a workflow network based on bug reports and extracted two features related to workflow network, while we construct a collaboration network based on the discussion among developers and extract a set of social network features. Only one feature, i.e., *the way to start participation (by commenting an existing report or by creating a new report via a bug report tool)*, is not considered in our approach, because we only regard a developer as a contributor of a project only if he/she has submitted a commit.

We use these similar features to Zhou and Mockus to build a random forest model and compare the AUC of this model with that built on all features. We apply the same evaluation approach as RQ1. The results of these prediction models are as shown in Table 24. We find that the average AUCs considering 1, 2, and 3 years as time-interval for *LTC* prediction are 0.712, 0.734, and 0.723, respectively. The improvements of the prediction model built on all of our proposed features are 6.7%, 7.6%, and 9.0%, respectively. We apply Wilcoxon signed-rank test [65] with Bonferroni correction [2] to investigate whether the models built on our features have improvement compare to those built on similar features to Zhou and Mockus. We also compute the Cliff's delta [10]. The improvements on AUC in three time interval settings are all statistically significant at the confidence level of 99% and of a large effect size. Hence, we believe that the prediction models built on our proposed features achieve better performance.

In our approach, we not only consider the contributors' first month activities, but also consider the contributors' experience and the historical activities of the project. This might be the reason that our approach achieves better performance than that of Zhou and Mockus. Some features that are related to experience belong to the top 10 important features that impact the effectiveness of our models, such as *user_history_followers*, *user_age*. We find that some features that indicate the history activities of a project, e.g., *before_repo_contributor_mean*, *before_repo_commits*, belong to the top 10 important features in our models. Additionally, the programming language belongs to the top 10 important features in the models of all three time interval settings. Thus, we believe that more information about developers and projects can improve the performance of prediction models. However, if a contributor is a pure newcomer to GITHUB and a project has few historical activities, our approach will have a similar performance to that of Zhou and Mockus.

5.2 Impact of 10-fold Cross-Validation

In the results we have presented, we keep the time order of joining the projects among developers to evaluate the results of all the prediction models (referred to as time-series cross-validation). As 10-fold cross-validation is widely used to evaluate the performance of prediction models, we consider whether there exist differences between the evaluation method used in our experiment and 10-fold cross-validation. In 10-fold cross validation, we randomly divide the dataset into ten folds. Of these ten folds, nine folds are used to train the classifier, while the remaining one fold is used to evaluate the performance. The class distribution in the training and testing dataset is kept the same as the original dataset to simulate real-life usage of the algorithm. We also use the parameter optimization method used to answer RQ1 and report the highest averaged AUCs with the most optimal parameter settings. Table 25 presents the averaged AUCs of time-series and 10-fold cross-validation of each prediction model considering different time intervals for *LTC*. For all classifiers, the AUCs using 10-fold cross-validation are better than those using time-series cross-validation. Random forest still has the best performance and its AUCs using 10-fold cross-validation are a bit larger than those using time-series cross-validation. In summary, we believe that the models built on our proposed features can still predict whether a developer will become a *LTC* effectively considering both time-series and ten-fold cross-validation.

5.3 Impact of Developer Productivity

Productivity is an important indicator for *LTCs* but the definition of *LTCs* in the experiment does not consider contributors' productivity. Hence, here we want to investigate whether another definition of *LTC* considering productivity would affect the performance of our proposed approach. More specially, given a time interval T, for the definition of *LTC* considering productivity, a *LTC* is a contributor who stay with the project for at least T (from his/her first

TABLE 23: The features used in the study of Zhou and Mockus [76] and our similar features.

Dimension	Their features	Our similar features
extent of involvement	the number of tasks (measured by the number of comments)	month_user_commits
	whether or not the first reported issue gets fixed	month_user_issue_events_closed
	whether or not starting participation from a comment for an existing issue or reporting a new issue	—
macro-climate	the number of product users	before_repo_watchers
	project sociality (measured by the number of participants in the first month)	month_repo_contributors
micro-climate	the number of peers in a developer's workflow network	degree_centrality closeness_centrality
	social clustering (measured by the amount of replication among the workflow networks of peers)	betweenness_centrality eigenvector_centrality clustering_coefficient
	the performance of peers (measured by the minimum number of issues in network)	month_repo_contributor_min
	the duration of time between the newcomers first action until somebody responds	month_user_issue_comments

TABLE 24: The averaged AUCs of random forest built on a subset of similar features to Zhou and Mockus and all features.

	1 Year	2 Year	3 Year
Similar features	0.712	0.734	0.723
All features	0.760	0.790	0.788

TABLE 25: AUCs of each prediction model in different time intervals for *LTC* using our evaluation method and 10-fold cross-validation

	Our evaluation method			10-fold		
	1 Year	2 Year	3 Year	1 Year	2 Year	3 Year
Naive Bayes	0.701	0.699	0.692	0.725	0.718	0.716
SVM	0.716	0.748	0.742	0.773	0.792	0.792
Decision Tree	0.722	0.750	0.743	0.630	0.497	0.529
KNN	0.667	0.731	0.710	0.750	0.768	0.751
Random Forest	0.768	0.802	0.783	0.814	0.840	0.841

commit) and who has productivity (measured via number of commits) exceeding Xth percentile among contributors with a tenure exceeding T. Table 26 presents the number of *LTC* and *non-LTC* for different settings of time interval (T) and productivity measurement (X). If we consider productivity to define *LTC*, the number of *LTC* would decrease because some contributors who are not very productive are excluded. However, the number of reduced contributors is not large and the ratios of *LTC* and *non-LTC* don't vary very much for the three time interval settings. Additionally, when the productivity measurement increases, the number of *LTC* does not vary very much. This indicates most contributors who stay with a project for a long time are considerably productive.

TABLE 26: The number of contributors for definition of LTC considering productivity.

	1 Year		2 Year		3 Year	
	#LTC	#non-LTC	#LTC	#non-LTC	#LTC	#non-LTC
Without Productivity	9,238	65,808	3,968	45,384	1,577	25,121
10th	7,994	67,052	3,637	45,715	1,489	25,209
20th	7,966	67,080	3,634	45,718	1,489	25,209
30th	7,860	67,186	3,613	45,739	1,480	25,218

We use the same approach in the RQ1 (see Section 3) to evaluate the performance of the prediction models on the new dataset considering contributors' productivity. Table 27 presents the averaged AUCs of each prediction model in three time intervals for *LTC*. For all prediction models, the AUCs increase when considering contributors' productivity but the improvement is small. Thus, we believe considering productivity to define *LTCs* does not affect our experiment results.

5.4 Impact of Feature Selection

Feature selection is widely used to improve performance of classifiers in previous studies [22]. In RQ3, we apply a feature selection method when we investigate the most important features. In this section, we would like to investigate whether automated feature selection methods can further improve the performance of our approach. For convenience, we denote our model without using automatic feature selection methods as Default. And we denote the feature selection method that is used in RQ3 as CRA, with C, R and A denotes correlation, redundancy and analysis, respectively.

We apply feature selection to all the features in five dimensions. And we use the longitudinal data setup, which is the same as RQ1, to evaluate the random forest models. In each fold, we first use CRA to remove correlated and redundant features for the training dataset. Then, we build a model using the preprocessed training dataset and evaluate the model on the testing dataset. Finally, we report the average AUC. We denote our model built with CRA applied

TABLE 27: AUCs of each prediction model in different time intervals for *LTC* without considering productivity and considering productivity.

	1 Year				2 Year				3 Year			
	Without Productivity	10th	20th	30th	Without Productivity	10th	20th	30th	Without Productivity	10th	20th	30th
Naive Bayes	0.701	0.718	0.718	0.718	0.699	0.715	0.715	0.714	0.692	0.700	0.700	0.700
KNN	0.667	0.684	0.684	0.685	0.731	0.740	0.741	0.741	0.710	0.717	0.717	0.717
Decision Tree	0.722	0.767	0.770	0.772	0.750	0.776	0.775	0.779	0.743	0.758	0.758	0.758
SVM	0.716	0.739	0.736	0.738	0.748	0.764	0.773	0.761	0.742	0.759	0.762	0.758
Random Forest	0.766	0.797	0.799	0.798	0.802	0.820	0.818	0.821	0.783	0.796	0.796	0.800

TABLE 28: AUCs of each prediction model built from all features and applied by feature selection.

	1 Year	2 Year	3 Year
Default	0.767	0.802	0.783
Default + CRA	0.777	0.804	0.784

as Default+CRA. Table 28 presents the average AUCs of Default+CRA models in comparison with Default models. We find the deltas between the performance of Default and Default+CRA models are small, i.e., 0.01, 0.002, 0.001 for time interval of 1, 2 and 3 years, respectively. Thus, we believe that the feature selection has little impact on the performance of our approach.

5.5 Impact of Project Popularity

We select popular projects (i.e., 1,000 most starred projects) in our earlier experiments. These popular projects might attract developers, which introduces a bias on the developers being *LTCs*. In this section, we want to investigate whether project popularity can affect the performance of our approach. We use another dataset collected by Munaiah et al. [43], which publish 800 engineered projects in GITHUB [12]. First, we remove 176 overlapping projects between these projects and the projects in our dataset. Then, we follow the steps in Section 2.1 to filter the remaining projects. Finally, we get 166 projects – the statistics of these projects are shown in Table 29. The mean and median star number of these projects are only 574.22 and 117, which are much less than those of the popular projects in our dataset. These less popular projects also have less contributors (34.14 vs. 120.54), commits (1648.58 vs. 3,697.33), issues (289.58 vs. 1,572.91) and pull requests (384.50 vs. 1,123.84). Table 30 presents the number of *LTC* and *non-LTC* contributors in different time intervals across these unpopular projects.

We construct two dataset using these unpopular projects: one only contains the contributors in the less popular projects (denoted as "Unpopular") and the other is that by combining contributors in the popular projects used in our earlier experiments and less popular projects (denoted as "Default + Unpopular"). We run random forest models on these dataset and report the average AUCs. Table 31 present the AUCs of the random forest models built on Default and Unpopular dataset. We find that AUCs of prediction models built on the "Unpopular" and the "Default + Unpopular" datasets are less than those built on the popular projects considering all time interval settings for *LTCs*. But all AUCs are larger than 0.7, which indicates a promising performance and the deltas are not very large. Thus, we believe that project popularity has not too much impact on the performance of our approach.

12. https://gist.github.com/nuthanmunaiah/23dba27be17bbd0abc40079411dbf066

TABLE 29: The statistics of unpopular projects

	Contributor	Commit	Issue	Pull Request	Star
Total	5633	272016	47781	63443	94747
Mean	34.14	1648.58	289.58	384.50	574.22
Median	13	335.5	48.5	57.5	117
Minimum	2	3	1	0	0
Maximum	341	16609	6139	3549	5213
Std.	48.63	2660.80	709.54	722.52	989.71

TABLE 30: The number of contributors in different time intervals across unpopular projects for definition of *LTC*

Year	#LTC	#non-LTC	Total
1	931	2,920	3,851
2	525	2,537	3,062
3	2,61	1,614	1,875

5.6 Evaluation using Matthews Correlation Coefficient

In this subsection, we evaluate our proposed approach using an additional measure, i.e., Mattews correlation coefficient (MCC) [37]. Unlike F1-measure, MCC takes account into all four quadrants of the confusion matrix, i.e., true positve (TP), false positives (FP), true negative (TN), and false negative (FN). Thus, it is generally regarded as a balanced measure which can be used even if the classes are highly imbalanced [7]. MCC is also widely used in previous software engineering studies [53], [54]. MCC is defined below:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (2)$$

The above formula returns a value between -1 and 1. A value of +1 indicates a perfect agreement between actuals and predictions, -1 indicates a perfect disagreement between

TABLE 31: AUCs of random forests built on unpopular and popular projects.

	1 Year	2 Year	3 Year
Default	0.768	0.802	0.783
Unpopular	0.734	0.793	0.729
Default + Unpopular	0.759	0.787	0.777

TABLE 32: MCCs of the prediction model in each time interval for *LTC*.

	1 Year	2 Year	3 Year
MCC	0.304	0.255	0.219

actuals and predictions, and 0 means the prediction may as well be random with respect to the actuals.

We use a threshold tuning approach to handle the high imbalance of our dataset. A threshold indicates a decision boundary to differentiate *LTC* from *non-LTC*. A classifier will classify a newcomer to be a *LTC* if its likelihood score is higher than the threshold. Usually, the default threshold is 0.5, however, the high imbalanced data causes a classifier to favor the majority class. Thus, we use a threshold tuning approach to automatically determine an imbalanced decision boundary using the training set [68]. Note that we did not use threshold tuning approach in the experiment, because the AUC would not be impacted by threshold.

The process of threshold tuning approach is: first, we randomly sample 20% instances of the training set (following a stratified sampling procedure) as the validation set, and use the remaining 80% instances of the training set as our new training set to build a classifier. Second, we use the built classier to evaluate the MCC score on the validation set by varying the threshold from 0.01 to 1.00, stepped by 0.01. Third, we determine the threshold that achieves the best MCC on the validation set. Finally, we use the new training set and the determined threshold to evaluate performance on our testing set.

We build random forest models using the same setting as RQ1 and report the average MCC for each setting of time interval for *LTC* prediction. Table 32 present average MCCs of the prediction model in each time interval for *LTC*. The average MCC of these prediciton models are 0.304, 0.255, and 0.219 in the time interval of 1, 2, and 3 years, respectively. Our proposed approach achieves similar MCCs to those in other prediction tasks such as defect prediction [54], [69], [75]. Due to highly imbalance nature of our dataset, we think the MCC results are reasonable.

5.7 Qualitative Analysis of Incorrect Predictions

In this section, we perform a qualitative analysis to understand the underlying reasons behind some incorrect predictions. The cases that we consider are the error cases in the first round of the 10-fold cross validation for *LTCs* predicted considering a time interval of 3 years. We have 49 error cases including 6 false positives and 43 false negatives. We focus on the five most important features as identified in RQ4, i.e., *user_history_followers*, *language*, *before_repo_commits*, *before_repo_contributor_mean*, and *month_repo_commits*.

For the false positive cases, we find that their features are very similar to true positive cases. For example, they are more popular than developers in the false negative cases, i.e., thay have 422 followers on average, while the average for the false negative cases is only 186. We randomly select two developers who are wrongly classified as *LTCs* of their respective project. The first developer submitted his initial commit to `slackhq/SlackTextViewController` on August 2014 and the second one submitted his initial commit to `kriasoft/react-starter-kit` also on August 2014. Both commits were the first commit of the respective project. We find that the first project has been "deprecated" since the project owner – a company named Slack – wants to focus on their internal projects. The second developer is still very active for other projects but not for `kriasoft/react-starter-kit` as the project does not grow much anymore. Note that our models are only learned based on developers activities. Based on this data, it is very difficult to predict a projects "fate" after a substantially long period of time time (i.e., 3 years) has passed. For these false positive cases, the corresponding projects are basically no longer under active development.

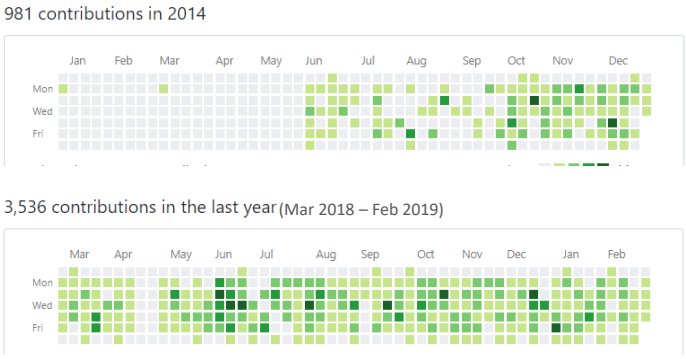

Fig. 1: The contributions of a developer in 2014 and now.

For the false negative cases, we also randomly select and analyze two developers who are wrongly classified as non-LTCs. The first developer submitted his first commit to the project `rapid7/metasploit-framework` on July 2014 and the second developer submitted his first commit to the project `owncloud/core` on the same month. When they started their first contribution, neither of them are popular (i.e., they only had 12 and 0 followers, respectively) nor active in GITHUB. Meanwhile, the two project already had a large number of commits (i.e., 17,471 and 25,292). Thus, our approach predicts them as non-LTCs. However, after that, both of them became more and more active; for example, before June 2014, the first developer made few contributions in GITHUB but after that he submitted a large number of contributions (see Figure 1). Thus, it is difficult to predict them as LTCs based on their activities in the first month as these developers behave "anomalously" as compared to others.

Although we cannot achieve zero false positive and negative cases, we have demonstrated that our approach can outperform the work of Zhou and Mockus by about 10%. An improvement of 10% on AUC have been considered substantial in many past work [14], [44], [49], [71].

5.8 Implications for Action

Retaining newcomers is important for the health and longevity of OSS projects. In this study, we proposed a data-driven approach to effectively predict whether newcomers will stay in a project. Maintainers of GITHUB projects can easily extract and extend our proposed features and build a prediction model for their own projects. Moreover, OSS

project maintainers can use features to monitor and manage the community contributing to the project. Based on our analysis results and feedback from the survey (see Section 4), we have several recommendations for OSS projects:

Retain experienced developers who show interest. We find that developers with larger *user_histroy_followers* are more likely to become LTCs. Additionally, in the survey, a respondent mentioned that *"how many other projects does a newcomer already contribute to"* is another factor that affects the chances that a newcomer will contribute for a long time. Another respondent said that *"In my experience developers become long time contributors of a project if they have another important project (work or personal) that depends on the project they contribute to."* It is important to attract and retain developers with these characteristics as they are usually experienced. When the first contribution of an experienced developer is detected, the project maintainers should notice and pay attention to the experienced newcomer. Some actions can be taken to improve the possibility of the contributors retention, such as communicating with the contributor to get their feedback, assigning tasks of interest and other similar actions.

Put special emphasis on encouraging more contributions in a newcomers first-month. We find that *month_user_commit*, a proxy of newcomers workload, is one important feature in the prediction models. Thus, after a newcomer makes a first contribution, project maintainers can encourage the contributor to make more by providing positive feedback and recommending more tasks of interest. As an alternative, a maintainer can assign an LTC to mentor a promising newcomer at least in the newcomers first month. One respondent mentions that having *"[m]entors that are already long-time contributors"* contributes positively to retaining newcomers. If a newcomer can be encouraged to make a considerable amount of contribution in their first month, there is a high likelihood that the newcomer would stay on as an LTC.

Improve the activeness of a projects community to retain newcomers. We find that *month_repo_contributor_mean*, which is an indicator of activeness of a project community, has a positive relationship with newcomers retention, i.e., the more work submitted by other contributors when a newcomer joins the project, the more likely the newcomer will stay in the project. In the survey, the majority of respondents who answer the last open question mentioned that the project community is an important factor that affects a newcomer to contribute for a long time. Thus, maintainers need to continue to innovate and drive the project with new requirements that can result in more activities in the project. This active community will in turn motivate the newcomers to stay and contribute more.

Different retention strategies may be appropriate for projects developed in different languages. Our results show that *language* is one of the most important features for the prediction models. The projects using C/C++, Java, and PHP have a much higher percentage of LTCs than those using Javascript, HTML, and Shell. Different from projects written in these scripting languages, we hypothesize that participation in projects written using C/C++, Java, and PHP may require more difficult technical and business knowledge [63], which increases contribution barriers. For these projects, maintainers should take some actions to lower the barrier for newcomers, such as providing more complete tutorials and documentation, improving the code quality to make the code easily understandable, and giving newcomers quick response to their queries. We find that respondents in the survey give many suggestions to retain newcomers, which are also applicable to these projects. For example, a respondent mentions the following ways: *"1) Proper documentation about how to contribute. 2) List of issues that could be solved by newcomers. 3) Active community members guiding newcomers. 4) Clear roadmap about new features..."*

On the other hand, for projects using Javascript, HTML and Shell, there are more contributors, but a lower percentage of LTCs. This might be because these scripting languages are easy to learn [63] so that many developers have the ability to make a contribution to the project using these languages. However, many of them are not willing to make continuous contributions to the projects. For these project, the project owner should be more inclusive. A respondent in the survey mentioned that *"The project owner sometimes thinks he knows more than a new contributor and this causes problem when they are wrong on a subject the newbie knows more about."* Project maintainers also need to pay particular attention to keeping the project community active. Thus, more developers will be attracted to submit their contributions

Projects in different stages of development should use different strategies to retain newcomers. We find that the popularity of a project may prevent newcomers from staying in the project, i.e., *before_repo_watchers* and *before_repo_issues* have a negative relationship with newcomers retention. Thus, for established projects, it is more difficult for newcomers to become LTCs. This finding highlights that maintainers of more popular projects should monitor the influx and departure of newcomers. If fewer newcomers do not want to contribute to the project for a long time, maintainers need to think about why the attractiveness of the project for newcomers has decreased and take action to retain newcomers. For example, project maintainers can communicate with newcomers frequently and giving timely responses to them. Our experiment results show that the features in the collaboration network have statistically significant differences and the effect size of the differences is small. Project maintainers can increase the "centrality" of newcomers by assigning more tasks to them and providing more guidance to them by commenting on and acknowledging their contributions. Some respondents suggested that the following are important factors to retain newcomers: *"Provide a community where they are encouraged to ask questions e.g. irc/slack and feel like they are part of a family"*, *"Maintainer's response on first issue/pull request"*, and *"Actionable comments during code review"*. For popular projects, more emphasis need to be put on these since it is easy to forget the newcomers and only focus on existing LTCs.

On the other hand, for new projects, newcomers are more likely to become LTCs. In the survey, a respondent said that *"Being an early contributor to project is strongest indicator of long term continued contribution"*. But project maintainers can use more approaches to retain newcomers at the beginning of the project. For example, a respondent

in the survey mentioned that some factors that worked for retaining newcomers include: *"(1) Code quality, (2) Explaining what the code fixes if it is a bug fix, (3) Explaining why a new feature is needed if submitted, and (4) Contributing hardware to the project to verify future code will still work."*

5.9 Threat to Validity

Threats to internal validity. To avoid the risk of selecting only an evaluation approach that provides strong results, we keep the time order of joining the projects among developers and divide the whole dataset into 10 windows. Then, we use the first n windows as the training data and the remaining $10 - n$ windows as the testing data. The results shows that the averaged AUCs with 1, 2, and 3 years as time-interval for *LTC* prediction are more than 0.75. We also compare our evaluation approach with 10-fold cross-validation and find that the average AUCs in 10-fold cross-validation are a bit more than those in our evaluation approach. This might be because the values of AUC increase when the number of training windows increases (see Table 7, 8, and 9), while the sizes of training data in 10-fold cross-validation are all the same and equal to 90% of contributors in our dataset. Thus, we believe our evaluation approach achieves similar performance with 10-fold cross-validation in terms of AUC. The criteria that we use to determine if a developer is an *LTC* is also a threat to validity. We only consider a developer as a contributor if he/she modifies source code of projects by submitting commits. However, developers can participate in the development of projects in other ways (e.g., reporting bugs, discussing in mailing list). The time interval requirements that we use to determine *LTCs* could also affect our findings. In this study, we determine newcomers that stay in a project for more than 1, 2, and 3 years to be *LTCs*. We have tried to mitigate this threat by considering multiple time-interval settings. Another threat to validity is that we use the right censoring method [40] to discard the developers who joins within time T (1, 2 and 3 years) of the data gathering time. In the future, we plan to use some survey analysis models to investigate these discarded data.

Threats to external validity. We select 917 GIHUB projects that use different programming languages and have different numbers of contributors. Although GITHUB contains much software development data, not all development activities of projects are recorded in GITHUB since they might use some other kinds of systems to manage project development and evolution, e.g., bug tracking systems, mail lists, etc. To mitigate this threat, we have excluded projects that do not have any issue data.

Threats to construct validity. We use AUC to evaluate the results of prediction models. AUC is widely used to evaluate the effectiveness of various software engineering studies [34], [36], [49]. Moreover, AUC, which is a threshold-independent measure, is recommended as a good measure to evaluate the performance of a classifier by many researchers [36], [58].

6 Related Work

Developer turnover, referred to as the phenomenon of continuous influx and retreat of developer in software development, has been studied for many years. Some researchers try to understand developers' motivations to join and reasons to leave OSS projects. Zhou and Mockus use the bug report data to predict whether a developer will become a long-time contributor [76]. Yu *et al.* found that the two most important factors to indicate developer turnover were the objective attribute of OSS projects and personal expectations [72]. Schilling *et al.* reported that the level of development experience and conversational knowledge were strongly associated with developer retention by analyzing the contribution behavior of former Google Summer of Code [51]. Hynninen *et al.* conducted a survey with developers and found that developer turnover could be an important manifestation of low commitment [26]. Sharma *et al.* built a linear logistic model that considers both the developer and project level factors and found past activity, developer role, project size and project age have important impacts on developer turnover [52]. Zanatta *et al.* conducted a study to identify barriers to newcomers contributing to software-crowdsourcing projects and found that lack of documentation is the first barrier that prevents newcomer to participate the projects [73]. Different to previous studies, our study uses a large scale number of OSS projects from GITHUB, considers cover more completed activities of developers, and uses data mining technique to predict developer turnover. Yamashita et al. also used a large number of projects from GITHUB to measure project characteristics from the perspective of developer attraction and retention [70]. Based on the developer migration in Github, they divided GITHUB projects into two categories: magnet and sticky projects. They find that $\sim$23% of developers remain in the same project that they contribute to and the larger projects are likely to attract and retain more developers. Comparing with this study, we extract many features in different dimensions based on developers' activities in GITHUB and predict whether a newcomer will contribute to a project for a long time.

The impact of developer turnover is also studied by many researchers. Hall *et al.* conducted a survey and found that developer turnover is crucial to the success of an OSS project. Developer turnover could also cause knowledge loss in software development team. Izquierdo-Cortazar *et al.* used the evolution of orphan lines of code lastly edited by a developer who left the team to measure the knowledge loss of the projects [27]. Fronza *et al.* proposed a wordle to visualize the level of cooperation of a team and mitigate the knowledge loss due to turnover [18]. Software quality is also related to developer turnover [16], [41]. Mockus finds that only leavers have relationship with software quality due to the loss of knowledge and experience [41]. On the contrary, Foucault *et al.* find that newcomers have a relationship with quality and leavers do not have such relationship [16].

With the huge data generated in modern software development, data mining techniques are applied to investigate developer turnover. Bao *et al.* [4] used monthly reports from two industry IT companies to extract 67 features along six dimensions. They found that the most effective classifier is random forest and the top three important features are the content of task report in monthly reports, the standard deviation of working hours, and the standard deviation of working hours of project members in the first month. Zhou

and Mockus [76] collected the bug reports from Mozilla and Gnome, which both contain more than 100,000 contributors. They built a linear logistic model based on 10 features from three dimensions: extent of involvement (contributors' activities), macro-climate (the overall of projects), and micro-climate (the environment of individual contributor). The reported average precision and recall of their model for predict who will stay in the project are 31.64% and 24.29%. Comparing to these studies, our data collected from GITHUB cover more developers' activities and more OSS projects. We also try different kinds of prediction models. Our results show that the random forest model achieves the best performance with more than 0.75 AUCs. We also find that the number of followers, the programming language and the averaged number of commits contributed byother developers when a newcomer joins a project are the three important features in three settings of time interval for *LTC*.

7 Conclusion & Future Work

In this paper, based on multiple kinds of data in software development from GITHUB, we apply data mining techniques to investigate whether newcomers will become *LTCs* of a project. Our data is constructed from GHTorrent and contains 917 projects. We use three time interval settings for *LTC* prediction: 1, 2, and 3 years, and get 9,238, 3,968, and 1,577 developers who become *LTCs*, respectively. Our study reveals that the most effective classifier (i.e., random forest) for predicting whether newcomers will become *LTCs* achieves more than 0.75 AUC for the three time interval settings. We find that the number of followers is the most important feature in all three time interval setting studied. We also find that the programming languages and average number of commits contributed by other developers when a newcomer joins a project also belong to the top 10 most important features in all three time interval setting for *LTC* prediction. In the future, we want to collect more developers' activities in OSS projects and further validate the effectiveness of our approach using more developers and projects.

References

[1] Scott-knott esd test. https://cran.r-project.org/web/packages/ScottKnottESD/ScottKnottESD.pdf.

[2] H. Abdi. Bonferroni and šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics*, 3:103–107, 2007.

[3] S. Baltes, R. Kiefer, and S. Diehl. Attribution required: Stack overflow code snippets in github projects. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 161–163. IEEE Press, 2017.

[4] L. Bao, Z. Xing, X. Xia, D. Lo, and S. Li. Who will leave the company?: a large-scale industry study of developer turnover by mining monthly work report. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 170–181. IEEE Press, 2017.

[5] K. Blincoe, J. Sheoran, S. Goggins, E. Petakovic, and D. Damian. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39, 2016.

[6] P. Bonacich. Power and centrality: A family of measures. *American journal of sociology*, 92(5):1170–1182, 1987.

[7] S. Boughorbel, F. Jarray, and M. El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6):e0177678, 2017.

[8] U. Brandes and C. Pich. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318, 2007.

[9] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] N. Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[11] S. Daniel, L. Maruping, M. Cataldo, and J. Herbsleb. When cultures clash: Participation in open source communities and its implications for organizational commitment. 2011.

[12] T. T. Dinh-Trong and J. M. Bieman. The freebsd project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, 2005.

[13] I. El Asri, N. Kerzazi, L. Benhiba, and M. Janati. From periphery to core: A temporal analysis of github contributors's collaboration network. In *Working Conference on Virtual Enterprises*, pages 217–229. Springer, 2017.

[14] Y. Fan, X. Xia, D. Lo, and A. E. Hassan. Chaff from the wheat: Characterizing and determining valid bug reports. *IEEE Transactions on Software Engineering*, 2018.

[15] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[16] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri. Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 829–841. ACM, 2015.

[17] L. C. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

[18] I. Fronza, A. Janes, A. Sillitti, G. Succi, and S. Trebeschi. Cooperation wordle using pre-attentive processing techniques. In *Proc. CHASE*, pages 57–64. IEEE, 2013.

[19] W. Fu, T. Menzies, and X. Shen. Tuning for software analytics: Is it really necessary? *Information and Software Technology*, 76:135–146, 2016.

[20] M. Goeminne and T. Mens. Evidence for the pareto principle in open source software activity. In *the Joint Porceedings of the 1st International workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability*, pages 74–82, 2011.

[21] G. Gousios. The ghtorrent dataset and tool suite. In *Proc. of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[22] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[23] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[24] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[25] J. R. Horgan, S. London, and M. R. Lyu. Achieving software quality with testing coverage measures. *Computer*, 27(9):60–69, 1994.

[26] P. Hynninen, A. Piri, and T. Niinimaki. Off-site commitment and voluntary turnover in gsd projects. In *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on*, pages 145–154. IEEE, 2010.

[27] D. Izquierdo-Cortazar. Relationship between orphaning and productivity in evolution and gimp projects. *ence, Eindhoven University of Technology, The Netherlands.*, page 6.

[28] O. Jarczyk, B. Gruszka, S. Jaroszewicz, L. Bukowski, and A. Wierzbicki. Github projects. quality analysis of open-source software. In *International Conference on Social Informatics*, pages 80–94. Springer, 2014.

[29] T. Jiang, L. Tan, and S. Kim. Personalized defect prediction. In *Proc. ASE*, pages 279–289. IEEE, 2013.

[30] Y. Jiang, B. Cukic, and T. Menzies. Can data transformation help in the detection of fault-prone modules? In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 16–20. ACM, 2008.

[31] S. Kim, E. J. Whitehead Jr, and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, 34(2):181–196, 2008.

[32] S. Koch and G. Schneider. Effort, co-operation and co-ordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[33] R. E. Kraut, P. Resnick, S. Kiesler, M. Burke, Y. Chen, N. Kittur, J. Konstan, Y. Ren, and J. Riedl. *Building successful online communities: Evidence-based social design*. Mit Press, 2012.

[34] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. Predicting the severity of a reported bug. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 1–10. IEEE, 2010.

[35] I. H. Laradji, M. Alshayeb, and L. Ghouti. Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58:388–402, 2015.

[36] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[37] B. W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[38] T. Mende. Replication of defect prediction studies: problems, pitfalls and recommendations. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, page 5. ACM, 2010.

[39] T. Menzies and M. Shepperd. Special issue on repeatable results in software engineering prediction, 2012.

[40] R. G. Miller Jr. *Survival analysis*, volume 66. John Wiley & Sons, 2011.

[41] A. Mockus. Succession: Measuring transfer of code and developer productivity. In *Proc. ICSE*, pages 67–77. IEEE, 2009.

[42] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[43] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22(6):3219–3253, 2017.

[44] J. Nam and S. Kim. Clami: Defect prediction on unlabeled datasets (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 452–463. IEEE, 2015.

[45] J. Nam, S. J. Pan, and S. Kim. Transfer defect learning. In *Proc. ICSE*, pages 382–391. IEEE Press, 2013.

[46] F. Rahman, D. Posnett, and P. Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 61. ACM, 2012.

[47] M. Ramos, M. T. Valente, R. Terra, and G. Santos. Angularjs in the wild: A survey with 460 developers. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 9–16. ACM, 2016.

[48] G. Robles, S. Koch, J. M. GonZÁlEZ-BARAHonA, and J. Carlos. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, pages 51–56. IET, 2004.

[49] D. Romano and M. Pinzger. Using source code metrics to predict change-prone java interfaces. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 303–312. IEEE, 2011.

[50] J. Saramäki, M. Kivelä, J.-P. Onnela, K. Kaski, and J. Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105, 2007.

[51] A. Schilling, S. Laumer, and T. Weitzel. Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 3446–3455. IEEE, 2012.

[52] P. N. Sharma, J. Hulland, and S. Daniel. Examining turnover in open source software projects using logistic hierarchical linear modeling approach. In *IFIP International Conference on Open Source Systems*, pages 331–337. Springer, 2012.

[53] M. Shepperd. How do i know whether to trust a research result? *IEEE Software*, 32(1):106–109, 2015.

[54] M. Shepperd, D. Bowes, and T. Hall. Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6):603–616, 2014.

[55] D. Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[56] I. Steinmacher, M. A. Gerosa, and D. Redmiles. Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective*, 2014.

[57] K. J. Stewart and S. Gosain. The impact of ideology on effectiveness in open source software development teams. *Mis Quarterly*, pages 291–314, 2006.

[58] C. Tantithamthavorn and A. E. Hassan. An experience report on defect modelling in practice: Pitfalls and challenges. In *In Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP'18)*, page To Appear, 2018.

[59] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 2018.

[60] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan. What are the characteristics of high-rated apps? a case study on free android applications. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 301–310. IEEE, 2015.

[61] R. Tourangeau, L. J. Rips, and K. Rasinski. *The psychology of survey response*. Cambridge University Press, 2000.

[62] E. Van Emden and L. Moonen. Java quality assurance by detecting code smells. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.*, pages 97–106. IEEE, 2002.

[63] G. Von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.

[64] S. Wehaibi, E. Shihab, and L. Guerrouj. Examining the impact of self-admitted technical debt on software quality. In *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'16)*, 2016.

[65] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[66] D. H. Wolpert and W. G. Macready. An efficient method to estimate bagging's generalization error. *Machine Learning*, 35(1):41–55, 1999.

[67] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[68] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang. Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology*, 61:93–106, 2015.

[69] X. Xuan, D. Lo, X. Xia, and Y. Tian. Evaluating defect prediction approaches using a massive set of metrics: An empirical study. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1644–1647. ACM, 2015.

[70] K. Yamashita, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi. Magnet or sticky? measuring project characteristics from the perspective of developer attraction and retention. *Journal of Information Processing*, 24(2):339–348, 2016.

[71] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang. Automating change-level self-admitted technical debt determination. *IEEE Transactions on Software Engineering*, 2018.

[72] Y. Yu, A. Benlian, and T. Hess. An empirical study of volunteer members' perceived turnover in open source software projects. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 3396–3405. IEEE, 2012.

[73] A. L. Zanatta, I. Steinmacher, L. S. Machado, C. R. de Souza, and R. Prikladnicki. Barriers faced by newcomers to software-crowdsourcing projects. *IEEE Software*, 34(2):37–43, 2017.

[74] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer. Categorizing bugs with social networks: a case study on four open source software communities. In *International Conference on Software Engineering*, pages 1032–1041, 2013.

[75] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou. Towards building a universal defect prediction model with rank transformed predictors. *Empirical Software Engineering*, 21(5):2107–2145, 2016.

[76] M. Zhou and A. Mockus. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2015.