Who Should Make Decision on this Pull Request? Analyzing Time-Decaying Relationships and File Similarities for Integrator Prediction

Jing Jiang[a], David Lo[b], Jiateng Zheng[a], Xin Xia[c], Yun Yang[a], Li Zhang[a*]

[a] *State Key Laboratory of Software Development Environment, Beihang University, Beijing, China*
jiangjing@buaa.edu.cn,zjt_james@163.com,ayonel@qq.com,lily@buaa.edu.cn
[b] *School of Information Systems, Singapore Management University, Singapore*
davidlo@smu.edu.sg
[c] *Information Technology, Monash University, Melbourne, VIC, Australia*
xin.xia@monash.edu

Abstract

In pull-based development model, integrators are responsible for making decisions about whether to accept pull requests and integrate code contributions. Ideally, pull requests are assigned to integrators and evaluated within a short time after their submissions. However, the volume of incoming pull requests is large in popular projects, and integrators often encounter difficulties in processing pull requests in a timely fashion. Therefore, an automatic integrator prediction approach is required to assign appropriate pull requests to integrators. In this paper, we propose an approach TRFPre which analyzes Time-decaying Relationships and File similarities to predict integrators. We evaluate the effectiveness of TRFPre on 24 projects containing 138,373 pull requests. Experimental results show that TRFPre makes accurate integrator predictions in terms of accuracies and Mean Reciprocal Rank. Less than 2 predictions are needed to find correct integrator in 91.67% of projects. In comparison with state-of-the-art approaches cHRev, WRC, TIE, CoreDevRec and ACRec, TRFPre improves top-1 accuracy by 68.2%, 73.9%, 49.3%, 14.3% and 46.4% on average across 24 projects.

Keywords:
Integrator prediction, Code review, Open source, GitHub

1. Introduction

Various open source software hosting sites, notably Github, provide support for pull-based development and allow developers to make contributions flexibly and efficiently [1]. In GitHub, contributors fork a project's main repository, and make their code changes independent of one another. When a set of changes is ready, contributors create and submit pull requests to main repository. Any developers can provide comments and exchange opinions about pull requests [2, 3]. Members of the project's core team (from here on, integrators) are responsible to inspect submitted code changes, identify issues (e.g., vulnerabilities), and decide whether to accept pull requests and merge these code changes into main repository [1]. Integrators may directly make decisions of pull requests, without leaving any comments.

Ideally, pull requests are assigned to integrators and evaluated within a short time after their submissions. However in practice, integrators often encounter difficulties in processing pull requests in a timely fashion. Gousios et al. made an exploratory qualitative study to understand integrators' work practices and challenges in GitHub [1]. They highlighted that integrators of popular projects mentioned that the volume of incoming pull requests was just too large, and integrators saw triaging pull requests as a challenge. Therefore, an automatic integrator prediction approach is required to assign appropriate pull requests to integrators.

There have been several studies about reviewer recommendation. Some previous works [4, 5, 6] analyzed past code review history to recommend code reviewers in a code review system Gerrit. In Gerrit, reviewers provide code-review scores and verified scores. Integrators in GitHub play a similar role as reviewers in Gerrit. It remains unknown whether these approaches are effective to predict integrators in GitHub. Other previous works [7, 8] mainly analyzed review comments and recommended commenters in GitHub. As described in subsection 2.1, some integrators directly make decisions of pull requests, without leaving any comments. Reviewer prediction based on comments fails to predict integrators for some pull requests.

In this paper, we propose an integrator prediction approach TRFPre which analyzes Time-decaying Relationships and File similarities based on previous pull request decisions. TRFPre rests on two key insights. The first is that integrators are not necessarily confined to developers who provide comments and exchange opinions. Some integrators do not leave any comments in pull requests. TRFPre studies integration decisions in previous pull requests, rather than comments in previous pull requests. The second is that developers come and go as they

*Corresponding author

please, resulting in high turnover [9]. TRFPre considers time-decaying relationships and time-decaying file similarities. Older pull requests are given lower weights.

The usage scenarios of our proposed tool are as follows:

Without Tool. Bob is an integrator in a large open source project team, and his main responsibility is to review pull requests submitted by other developers. Without our tool, he browses pull request lists, and selects pull requests which are appropriate for him. Then, he inspects code, detects bugs, and discusses with other developers. Since some popular projects receive many pull requests, pull request selection costs much time, and reduces Bob's time on code review. This problem becomes more serious for some integrators, who are volunteers and have limited time in open source software projects.

With Tool. Bob's project adopts our tool. Our tool assigns appropriate pull requests to Bob. Then, Bob directly inspects pull requests and reviews code changes. Since our tool saves Bob's time on pull request selection, Bob has more time on code review and evaluates more pull requests.

In an effort to demonstrate the effectiveness of our approach, we collected datasets from GitHub. In total, we analyze 24 projects and 138,373 pull requests. We measure the performance of approaches in terms of top-1 and top-2 accuracies, and Mean Reciprocal Rank (MRR) [6]. The experimental results show that TRFPre makes accurate integrator predictions in terms of accuracies and MRR. Less than 2 predictions are needed to find correct integrator in 91.67% of projects. In comparison with the state-of-the-art approaches cHRev [4], WRC [5], TIE [6], CoreDevRec [10] and ACRec [8], TRFPre improves top-1 accuracy by 68.2%, 73.9%, 49.3%, 14.3%, and 46.4% on average across 24 projects.

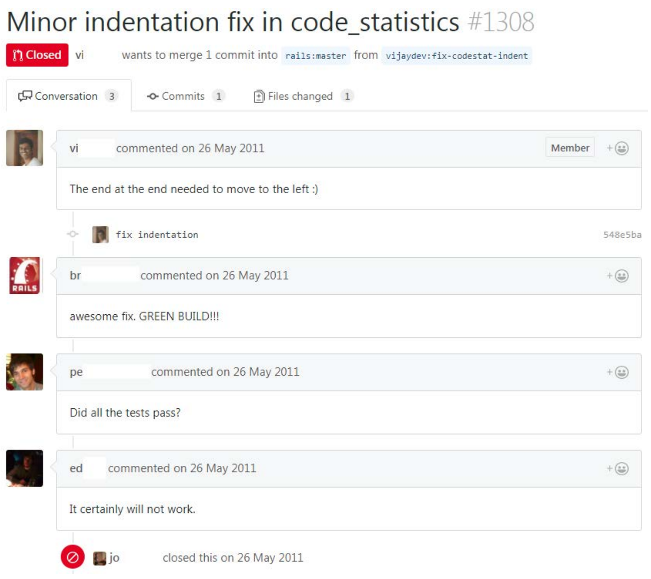
The main contributions of this paper are as follows:

- We propose an integrator prediction approach TRFPre, which analyzes time-decaying relationships and file similarities based on previous pull request decisions.
- We evaluate TRFPre based on a broad range of datasets. Results show that TRFPre outperforms cHRev, WRC, TIE, CoreDevRec and ACRec by substantial margins.

The reminder of the paper is organized as follow. Section 2 presents a background of pull request evaluation, data collection and basic data statistics. Section 3 presents our integrator prediction approach TRFPre. Section 4 presents an empirical evaluation of the approach. Section 5 discuss ground truth and feature importance. Section 6 discusses threats to validity, and Section 7 discusses related works. Finally, Section 8 concludes this paper.

2. Background and Data Collection

In this section, we begin by providing background information about contribution evaluation process in GitHub, and contrast it with the process in Modern Code Review (MCR). Then, we introduce how our datasets are collected, and report statistics of our datasets.



Minor indentation fix in code_statistics #1308

Closed vi wants to merge 1 commit into rails:master from vijaydev:fix-codestat-indent

Conversation 3 Commits 1 Files changed 1

vi commented on 26 May 2011 Member

The end at the end needed to move to the left :)

fix indentation 548e5ba

br commented on 26 May 2011

awesome fix. GREEN BUILD!!!

pe commented on 26 May 2011

Did all the tests pass?

ed commented on 26 May 2011

It certainly will not work.

jo closed this on 26 May 2011

Figure 1: An example of pull request evaluation

2.1. *Pull Request Evaluation*

GitHub is a web-based hosting service for software development repositories in different areas, such as cloud computing [11, 12]. In GitHub, contributors fork a project's main repository, and make their code changes independent of one another. When a set of changes is ready, contributors create and submit pull requests to main repository. Any developers can leave comments and exchange opinions about pull requests. Developers freely discuss whether code style meets the standard [13], whether repositories require modification, or whether submitted codes have good quality [2]. According to comments, contributors may modify codes. Finally, integrators inspect submitted code changes, and decide whether to integrate these code changes into main repository or not [1]. Integrators act as guardian for project quality [14].

In GitHub, integrators decide whether to integrate code changes into repositories or not. Any developers can provide comments and discuss pull requests. Integrators may provide comments and ask contributors to modify codes. However, providing comments is not necessary for integrators. Integrators may directly decide to accept or reject pull requests, without leaving any comments.

To illustrate the process, Figure 1 shows an example of a pull request with number 1308 in project *rails*[1]. Firstly, a contributor *vi**** modified codes and submitted a pull request. In order to protect developers privacy, we only show part of characters in developers names. Secondly, developers *br****, *pe**** and *ed**** leaved comments and discussed the pull request. Finally, an integrator *jo**** decided to reject this pull request. In this example, integrator *jo**** directly made decision, and did not provide any comments.

Let us now contrast the above process with Modern Code Review (MCR). In MCR, typically a developer submits a code

[1]https://github.com/rails/rails/pull/1308

Table 1: Basic information of three pull requests

Pull request	Contributor	Integrator	Creation time	Files changed
494	vi***	jo***	2011-5-11	railties/lib/rails/commands/runner.rb
1293	ja**	jo***	2011-5-25	railties/lib/rails/paths.rb
1308	vi***	jo***	2011-5-26	railties/lib/rails/code_statistics.rb

Table 2: Percentage of pull requests which their integrators do not provide comments

Project	Percentage of pull requests
rails	26.93
commcare-hq	42.51
tgstation	30.03
symfony	30.23
cocos2d-x	71.55
core	4.43
Baystation12	39.58
joomla-cms	26.7
app	29.41
metasploit-framework	26.85
bootstrap	26.84
dmd	35.66
cdnjs	24.57
zendframework	37.93
angular.js	23.79
cakephp	22.2
puppet	36.66
brackets	16.34
scala	20.35
ipython	18.01
sympy	12.71
node-v0.x-archive	14.68
wet-boew	66.3
katello	10

Table 3: Basic Statistics of projects.

Project	# Pull requests	# Integrators	Date of First Pull Request
rails	14,237	46	2010-9-2
commcare-hq	11,572	31	2012-3-14
tgstation	10,327	35	2012-1-21
symfony	9,222	17	2010-9-1
cocos2d-x	8,939	13	2010-11-20
core	7,509	45	2012-8-25
Baystation12	7,062	21	2011-11-7
joomla-cms	6,252	27	2011-9-28
app	5,762	99	2012-8-23
metasploit-framework	5,279	46	2011-11-10
bootstrap	4,948	12	2011-8-19
dmd	4,923	14	2011-1-26
cdnjs	4,914	12	2011-2-26
zendframework	4,762	15	2010-9-4
angular.js	4,526	32	2010-9-8
cakephp	4,512	21	2010-9-5
puppet	4,105	82	2010-9-28
brackets	4,006	42	2011-12-8
scala	3,644	19	2011-12-1
ipython	3,628	14	2010-9-17
sympy	2,915	30	2010-9-1
node-v0.x-archive	2,374	29	2010-8-31
wet-boew	1,703	11	2012-5-25
katello	1,252	24	2012-4-12
Total	138,373	737	

change to a code review system, e.g., Gerrit. Then, reviewers discuss the change, leave comments and provide suggestions. Next, the developer improves the change according to comments. Reviewers provide code-review scores and verified scores. The code change will be integrated to the main repository when it receives a code-review score of +2 (Approved) and a verified score of +1 (Verified) in Gerrit [13]. In GitHub, integrators directly decide to accept or reject pull requests, and do not give any scores.

The example given in Figure 1 is not a sole example. Table 2 describes the percentage of pull requests in our dataset (described in Section 2.2.) for which the integrators do not provide comments. In project cocos2d-x, 71.55% of pull requests integrators do not provide any comments. In 75% of the projects that we investigate in this work (18 out of 24), more than 20% of pull requests integrators do not provide any comments.

2.2. Data Collection and Statistics

GitHub provides access to its internal data through an API. It allows us to access rich collection of OSS projects, and provides valuable opportunities for research. We gather information through GitHub API and create datasets of projects.

Unpopular projects receive few pull requests, and do not need integrator prediction. In data collection, we choose popular projects, because they receive many pull requests and need integrator prediction. We obtain a list of projects from previous work [15]. Vasilescu et al. studied effects of continuous integration on software quality and productivity outcomes [15], and made their research projects public[2]. We sort their projects by the number of pull requests. Then we select 29 projects with the most number of pull requests.

We collected pull requests of these 29 projects through GitHub API in August 2016. We sent queries to GitHub API, received its replies, and extracted data from project creation time to July 31, 2016. For each pull request, we crawled its ID, the contributor who submitted it, the creation time, the close time, paths of modified files and the developer who closed the pull request. The pull request could be closed by its contributor or an integrator. We ignored pull requests closed by their contributors, because their final decisions were not made by integrators and did not need integrator prediction. For remaining pull requests, we collected their integrators as developers who closed pull requests. The contributor wrote a title to summarize the modification of a pull request, and we gathered this text information. We also collected comments of pull requests, including their submission time and commenters.

We cannot directly collect integrator set through GitHub API. Only integrators are granted the privilege of making decisions of pull requests submitted by others [16]. We analyze pull

[2]https://github.com/Yuyue/pullreq_ci/blob/master/all_projects.csv

requests and build a set of integrators for each project. More specifically, if developers ever make decisions on pull requests submitted by others, they are considered as integrators. This integrator set includes active integrators who really work in projects. We collect datasets of 29 projects, and find that 5 projects have less than 10 integrators. We exclude these 5 projects, and only study 24 projects with more than 10 integrators, which really face integration prediction problem.

Table 3 presents statistics of collected data. The columns correspond to project name (Project), the number of pull requests (# Pull requests), the number of integrators (# Integrators), and date of first pull request. We ignored pull requests closed by their contributors, because their final decisions were not made by integrators and did not need integrator prediction. Though we collected datasets later than previous work [15], our datasets only have pull requests closed by integrators, and the number of pull requests in Table 3 may be smaller than previous work [15]. Some projects begin to use pull requests some time after project creation. The date of first pull request shows start time of code review based on pull requests. In total, our datasets include 138,373 pull requests and 737 integrators.

3. Proposed Approach

In this section, we describe our method TRFPre which analyzes Time-decaying Relationships and File similarities to predict integrators. We first introduce motivation and framework of our approach, respectively. Then we describe features and classifiers of TRFPre to solve integrator prediction problem.

3.1. Motivation

Table 1 describes basic information of pull requests with numbers 494, 1293 and 1308 in project *rails*. Decisions for these three pull requests were made by integrator *jo****. From these three pull requests, we can observe the following:

First, previous evaluation relationships are good indicators to predict appropriate integrators. For example, pull request with number 494 was created by contributor *vi****, and it was decided by integrator *jo****. Then contributor vi*** created another pull request with number 1308, and integrator *jo**** also made decision for this pull request.

Second, changed files could also be helpful to predict suitable integrators. An integrator is likely to evaluate pull requests which modify changed files located in similar locations. For example, these three pull requests all modify changed files which belong to folder railties/lib/rails, and they are all decided by the same integrator *jo****.

Third, an integrator who makes decisions on pull requests at a particular time point is likely to evaluate other pull requests in its near future. These three pull requests were created within a 15-day interval (May 11 - May 26). We refer to this property as temporal locality of integrators.

The above observations tell us that relationships, file similarities and temporal locality may be helpful for integrator prediction. Therefore, we use time-decaying relationships and file similarities to build our integrator prediction approach TRFPre.

3.2. Method Framework

As shown in Figure 2, the entire framework contains two phases: a model construction phase and a prediction phase. In model construction phase, a composite model TRFPre Composer is built from historical pull requests with known integrators. In prediction phase, the model is used to predict integrators for new pull requests.

In model construction phase, TRFPre first collects various information from a set of training pull requests with known integrators. We independently extract time-decaying relationships (Step 1) and time-decaying file similarities (Step 2) from crawled information. We describe detailed definitions and why we choose these features in subsections 3.3 and 3.4. According to pull request features and their integrators, we build random forest classifier which decides probabilities of integrators (Step 3). Random forest classifier constructs a multitude of decision trees for classification. Random forest classifier corrects for decision trees' habit of overfitting to their training set. TRFPre independently builds another naive bayes classifier based on training pull requests with known integrators (Step 4). Naive bayes classifier assumes independence among predictor variables, and determines response based on the maximum log-likelihood. Naive bayes classifier requires a small number of training data to estimate parameters necessary for classification. Next, these two classifiers are blended to construct TRFPre composer (Step 5). Random forest classifier and naive bayes classifier have different advantages in classification. The combination takes advantages of both classifiers. In subsection 4.6, results show that the combination of two classifiers achieves better performance than a single classifier. We implement two classifiers on top of the tool Weka[3].

In prediction phase, we use TRFPre to predict whether a pull request is likely to be evaluated by a specific integrator. TRFPre first extracts time-decaying relationships (Step 6) and time-decaying file similarities (Step 7). Then, it processes these features into the random forest classifier built in the model construction phase (Step 8). It also processes features into the naive bayes classifier built in the model construction phase (Step 9). These two classifiers will independently output two probabilities of integrators, and these probabilities are combined by leveraging the TRFPre composer constructed in the model construction phase (Step 10). Integrators with the highest probabilities are predicted.

3.3. Time-decaying Relationship

Developer social networks are often used in quality prediction, defect prediction, bug triage and bug fixing [17]. Since integration process is mainly a human process, we consider socio-technical aspect of collaboration to predict integrators. The basic intuition is that developers who recently evaluate a contributor's pull requests may be more likely to be appropriate integrators.

We use decision histories to build relationships between integrators and contributors. For a pull request P_j ($P_j \in PSet_i$), its

[3]http://www.cs.waikato.ac.nz/ml/weka/

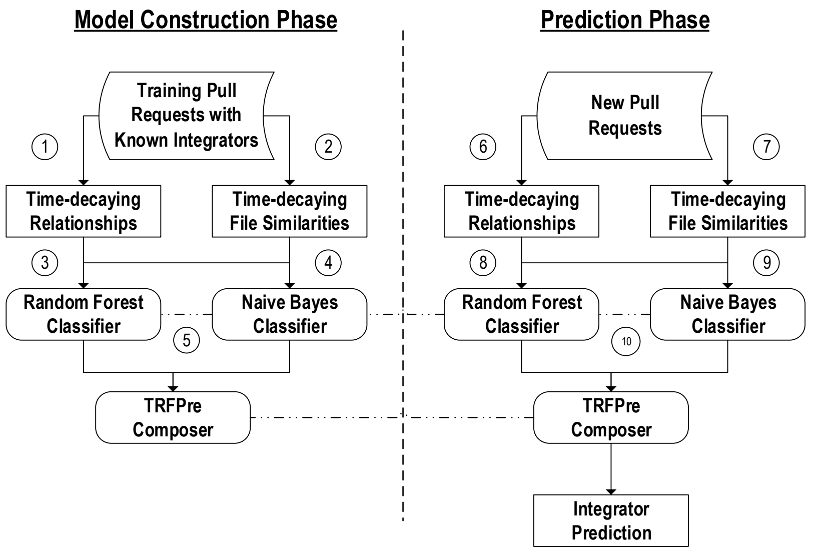


Figure 2: Overall framework of TRFPre

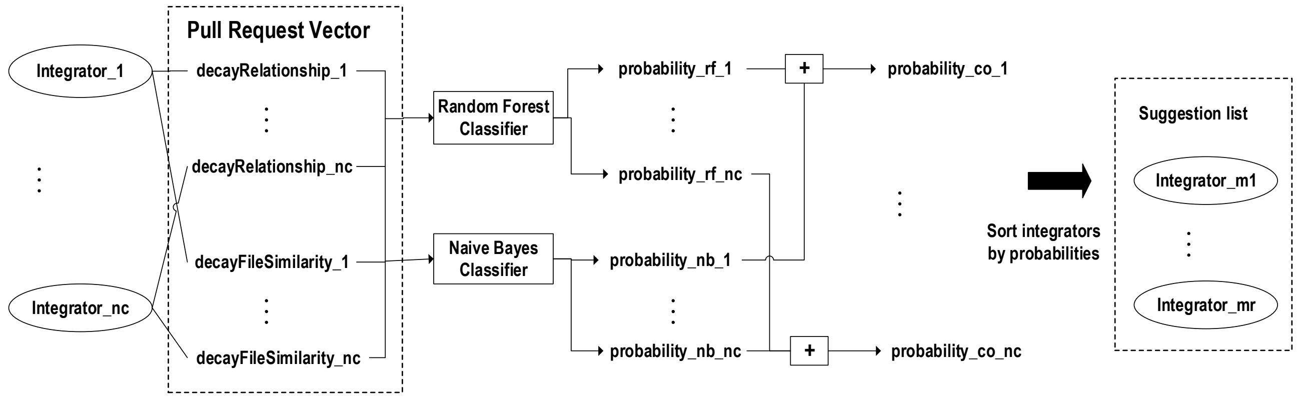


Figure 3: Integrator ranking process

creation time is t_j. Given a new pull request P_{new}, its contributor is defined as O_{new}. $OSet_{new}$ includes pull requests which are submitted by contributor O_{new} before the new pull request P_{new}. Let denote the i^{th} integrator as D_i. $PSet_i$ includes pull requests on which integrator D_i has made decision on within λ days before new pull request P_{new}. λ is used to describe the length of temporal window, and we exclude pull requests which are created more than λ days before new pull request. The intersection $OSet_{new} \bigcap PSet_i$ includes pull requests, which are submitted by contributor O_{new} and integrator D_i has made decision on within λ days before new pull request P_{new}. This intersection $OSet_{new} \bigcap PSet_i$ may reflect the integrator D_i's interest in the contributor O_{new}. If integrator D_i often make decision on pull requests submitted by contributor O_{new} , integrator D_i may be familiar with this contributor and like to make decisions on his or her pull requests. For a new pull request P_{new}, we compute *time-decaying relationship* between integrator D_i and contributor O_{new}, denoted as *decayRelationship*(P_{new}, D_i), as follows:

$$\begin{aligned} &decayRelationship(P_{new}, D_i) \\ &\quad = \sum_{(P_j \in (OSet_{new} \bigcap PSet_i))} (t_{new} - t_j)^{-1} \end{aligned} \tag{1}$$

We use $(t_{new} - t_j)^{-1}$ to measure pull requests' weights. We compute time interval (measured in days) between t_{new} and t_j. Any pull requests in intersection of $OSet_{new}$ and $PSet_i$ are created before new pull request P_{new}, and t_{new} is larger than t_j. Therefore, $(t_{new} - t_j)$ is larger than 0. Pull requests in more recent time have higher weights. If an integrator has made decisions on many pull requests recently submitted by a contributor, the integrator has close relation with this contributor. If an integrator never makes decisions on pull requests submitted by a contributor within λ days before new pull request, *decayRelationship*(P_{new}, D_i) is set as 0.

3.4. Time-decaying File Similarity

Previous work [18] observed that files located in similar paths would be reviewed by similar experienced code reviewers. Integrators are familiar with some files, and they may make decisions on same files or files in similar locations. We use longest common prefix in previous work [18] to compute file similarity, and then add time-decaying function to compute time-decaying file similarity.

In order to compute file distance between pull requests, we need to define file similarity at first. Given two files $file_a$ and $file_b$, path distance *fileDistance*$(file_a, file_b)$ is calculated as follow:

$$\begin{aligned}&pathDistance(file_a, file_b)\\&= \frac{prefixLength(file_a, file_b)}{maxLength(file_a, file_b)}\end{aligned} \tag{2}$$

The length of a file is the number of substrings in file path. We take pull request with number 1308 in Table 1 as an example. This pull request has 1 modified file, namely "railties/lib/rails/code_statistics.rb". The file path include 4 substrings, including "railties", "lib", "rails" and "code_statistics.rb". Therefore, the length of this file is 4. *maxLength*$(file_a, file_b)$ is the maximum value of lengths of file $file_a$ and file $file_b$. Another pull request with number 1293 has 1 file "railties/lib/rails/paths.rb", and its length is also 4. When we compare 2 files in pull requests with numbers 1308 and 1293, their maximum value of lengths is 4.

As described in previous work [18], the longest common prefix length *prefixLength*$(file_a, file_b)$ is the number of the longest consecutive path components that appear in the beginning of both file paths. We still take above 2 files as an example. "railties", "lib", "rails" are the longest consecutive substrings that appear in the beginning of both file paths. Therefore, the number of common substrings is 3. The path distance is computed as the ratio of the longest common prefix length to the max length. The path distance between above 2 files is $\frac{3}{4}$.

Next, we define file distance between pull requests. Let us consider that files in pull request P_j form file set $FSet_j$. $FSet_{new}$ includes files in pull request P_{new}. For a new pull request P_{new} and a former pull request P_j, we compute pull request distance, denoted as *prDistance*(P_j, P_{new}), as follows:

$$\begin{aligned}&prDistance(P_{new}, P_j)\\&= \frac{\sum_{\substack{file_a \in FSet_j\\ file_b \in FSet_{new}}} pathDistance(file_a, file_b)}{\mid FSet_j \mid \times \mid FSet_{new} \mid}\end{aligned} \tag{3}$$

In GitHub, file modification is required to submit pull requests. Therefore, both $\mid FSet_j \mid$ and $\mid FSet_{new} \mid$ are lager than 0. We sum up the file similarity of all possible pairs of files from pull requests P_{new} and P_j, and finally compute average file similarity by dividing the sum with the number of possible pairs. The pull request distance is the average file similarity of all possible pairings of files.

In previous work [18], all pull requests have the same weights. Different from previous work [18], we give higher weights to pull requests in more recent time. We compute time-decaying file similarity of the integrator D_i, denoted as *decayFileSimilarity*(P_{new}, D_i), as follows:

$$\begin{aligned}&decayFileSimilarity(P_{new}, D_i)\\&= \sum_{P_j \in PSet_i} prDistance(P_{new}, P_j) * (t_{new} - t_j)^{-1}\end{aligned} \tag{4}$$

As described in equation 1, $(t_{new} - t_j)$ is larger than 0. An integrator will have high time-decaying file similarity, if this integrator recently makes decisions on pull requests with similar files as new pull request. As described in subsection 3.3, $PSet_i$ includes pull requests on which integrator D_i has made decision on within λ days before new pull request P_{new}. We exclude pull requests which are created more than λ days before new pull request. If an integrator never makes decisions on pull requests submitted by a contributor within λ days before new pull request, *decayFileSimilarity*(P_{new}, D_i) is set as 0.

3.5. TRFPre Composer

Figure 3 shows integrator ranking process for a new pull request. We represent each pull request as a vector. Each feature is an element in the vector. For a pull request P_{new} submitted by the contributor O_{new}, we compute values of time-decaying relationship $decayRelationship\_1$ and time-decaying file similarity $decayFileSimilarity\_1$ for integrator D_1. We also compute time-decaying relationships and time-decaying file similarities for other integrators.

The number of integrators in a project is denoted as nc. We combine all features together to generate an integrator vector, which includes $2 \times nc$ elements for each pull request. For a new pull request P_{new}, element 1 in integrator vector is the time-decaying relationship $decayRelationship\_1$ between integrator D_1 and contributor O_{new}; element nc in integrator vector is the time-decaying relationship $decayRelationship\_nc$ between integrator D_{nc} and contributor O_{new}; element $nc + 1$ in integrator vector is the time-decaying file similarity $decayFileSimilarity\_1$ of integrator D_1; element $2nc$ in integrator vector is the time-decaying file similarity $decayFileSimilarity\_nc$ of integrator D_{nc}.

Random forest classifier and naive bayes classifier independently assign labels (in our case: the integrator) to a data point (in our case: a pull request) with a certain probability. Given a new pull request P_{new}, for each integrator D_i, random forest classifier and naive bayes classifier output probabilities $probability\_rf(P_{new}, D_i)$ and $probability\_nb(P_{new}, D_i)$ that a new pull request P_{new} will be assigned to integrator D_i. Then TRFPre combines probabilities for each integrator D_i as follows:

$$\begin{aligned}&probability\_co(P_{new}, D_i)\\&= probability\_rf(P_{new}, D_i) * \gamma\\&+ probability\_nb(P_{new}, D_i) * (1 - \gamma)\end{aligned} \tag{5}$$

In above equation, γ is between 0 and 1. If $\gamma = 0$, TRFPre approach only uses naive bayes classifier. If $\gamma = 1$, TRFPre approach only uses random forest classifier. The value of γ can be empirically determined. By default, we set the value of γ as 0.7. We discuss setting reason in subsection 4.6.

Given a new pull request P_{new}, for each integrator D_i, we compute composite probability. Integrators with the highest composite probabilities are predicted for new pull request.

4. Evaluation

In this section, we present results of our evaluation for proposed approach. The aim of this study is to investigate the effectiveness of TRFPre approach in providing integrator prediction solutions. We first present research questions, evaluation procedure and evaluation metrics. We then present our experiment results that answer these research questions. The experimental environment is a windows server 2012, 64-bit, Intel(R) Xeon(R) 1.90 GHz server with 24GB RAM.

4.1. Research Questions

We are interested to answer following research questions:
RQ1: *How accurate is TRFPre in predicting integrators for pull requests?*

We propose TRFPre to find appropriate integrators for pull requests. We aim to evaluate the performance of our method in terms of accuracies and MRR.

RQ2: *How do model construction time, prediction time, accuracies, MRR and of TRFPre, cHRev [4], WRC [5], TIE [6], CoreDevRec [10] and ACRec [8] compare in predicting integrators?*

In terms of model construction time and prediction time, accuracies and MRR, we compare TRFPre with reviewer prediction methods [5, 6], integrator prediction method [10] and commenter recommendation method [4, 8]. We mainly We describe these methods' supervised learning technique, proposed metrics, pros and cons in Table 4.

cHRev [4] mainly analyzes reviewers' frequency, workdays, and recency. Reviewers who recently and frequently provide comments to the same files of new reviews are recommended. cHRev does not have model training process, and thus does not have time overhead of model construction phase. As show in Table 2, more than 20% of pull requests' integrators do not provide any comments in 18 (75%) projects. However, cHRev analyzes review comments while many pull requests in GitHub are not commented.

In order to predict reviewers, WRC [5] analyzes past reviews with the same files. WRC computes weighted review count which specifies the review experience a developer has with a specific file at the time a specific review is made. WRC recommends reviewers who are recently active in reviewing the same files as new reviews. Similar to cHRev, WRC does not have time overhead of model construction phase. However, WRC does not consider file similarities. If previous reviews have similar but different files as new review, these reviews are not considered in reviewer recommendation.

To address the challenge of assigning suitable reviewers to changes, TIE [6] integrates an incremental text mining model which analyzes the textual contents in a review request, and a similarity model which measures the similarity of changed file paths and reviewed file paths. The text mining model is based on naive bayes classifier, while similarity model does not need model construction. TIE analyzes previous reviews whose upload time is within the past 100 days before new review. However, previous reviews uploaded within 100 days have the same weights in reviewer recommendation.

In order to predict integrators, CoreDevRec [10] constructs a file path substring vocabulary, and builds path substring features. CoreDevRec independently considers 4 social connection features and 6 activeness features, such as following relationship and the close time of the latest pull request. CoreDevRec uses support vector machines (SVM) to analyze these features and make prediction. In the majority of projects, the number of substrings in file path vocabulary is more than 4,000. Therefore, CoreDevRec has many substrings in model training and long model construction time. The large number of substrings may also introduce noise that affects its accuracy.

ACRec [8] proposes an activeness based approach to recommend commenters for pull requests. ACRec simply recommends developers who recently and frequently provide comments. ACRec does not have time overhead of model construction phase. However, ACRec does not consider pull requests which do not have comments.

RQ3: *What are appropriate configuration settings in integrator prediction?*

TRFPre uses time-decaying function $(t_{new} - t_j)^{-1}$ to measure pull requests' weights. Pull requests submitted in more recent time are given higher weights. We would like to investigate benefit of time-decaying function in integrator prediction.

Furthermore, the parameter γ is used to assign weights of prediction likelihoods of naive bayes classifier and random forest classifier. By default, we set parameter γ as 0.7. We would like to investigate accuracies and MRR of TRFPre for various γ values.

In equations1 and 4, $PSet_i$ includes pull requests on which integrator D_i has made decision on within λ days before new pull request P_{new}. λ is used to describe the length of temporal window, and we exclude pull requests which are created more than λ days before new pull request. λ is set as no time limit by default. We explore how the setting of λ affects performance of TRFPre.

4.2. Evaluation Procedure

In order to simulate the usage of methods in practice, we sort all pull requests in chronological order of their creation time. Then pull requests created in the same month are put into a group.

Next, we use these groups to build training sets and testing sets. For the N^{th} round, pull requests created in N months after the first pull request are used to build a training dataset, and pull requests created in the $(N + 1)^{th}$ month are used to build a testing dataset. For example in the first round, the training set is built by pull requests created in the first month after the first pull request, and the testing set is built by pull requests created in the second month after the first pull request. Then in the second round, we build training set using pull requests created within two months after the first pull request, and build testing set using pull requests created in the third month. If no pull requests are submitted in a month, we ignore this month.

Table 5: Number of rounds

Project	# Rounds
rails	69
commcare-hq	47
tgstation	42
symfony	67
cocos2d-x	65
core	46
Baystation12	56
joomla-cms	57
app	45
metasploit-framework	56
bootstrap	59
dmd	66
cdnjs	65
zendframework	59
angular.js	67
cakephp	66
puppet	64
brackets	55
scala	55
ipython	67
sympy	67
node-v0.x-archive	57
wet-boew	50
katello	51

We use the training set and testing set to compute the performance of TRFPre in each round, and then compute the average accuracies and MRR of all pull requests. This setup ensures that only past pull requests are used to build prediction model.

Table 3 shows date of first pull request in projects. We analyze pull requests between the date of first pull request and July 31, 2016. Table 5 shows the number of rounds in projects. All projects have more than 40 rounds, and 10 projects have more than 60 rounds.

4.3. Evaluation Metrics

In order to evaluate our method, we use accuracy and Mean Reciprocal Rank (MRR). These metrics are commonly used in evaluation of reviewer prediction approaches [5, 6, 18, 19].

We evaluate performance of integrator prediction with accuracy of top m predicted integrators, as described in initial study [6]. The definition of the top m accuracy is as follows:

$$Accuracy_m = \frac{\sum_{pr \in PR} IsCorrect(pr)}{|PR|} \tag{6}$$

where m is the number of predicted integrators and PR is testing set of pull requests. $|PR|$ is the number of pull requests in testing set. If one of top m predicted integrators really make decision on a pull request, prediction is correct and $IsCorrect(pr)$ function returns value of 1; otherwise, prediction is incorrect and $IsCorrect(pr)$ function returns value of 0. Accuracy describes the percentage of pull requests which are correctly assigned to integrators. We choose m value to be 1 and 2 in experiments.

According to previous work [18], Mean Reciprocal Rank (MRR) measures average value of reciprocal ranks of correct integrators in a prediction list. The reciprocal rank of a query response is multiplicative inverse of rank of the first correct answer. The definition of MRR is as follow:

$$MRR = \frac{\sum_{pr \in PR} \frac{1}{rank(candidates(pr))}}{|PR|} \tag{7}$$

The $rank(candidates(pr))$ returns rank value of actual integrator in prediction list $candidates(pr)$. If prediction list does not include actual integrator, the value of $\frac{1}{rank(candidates(pr))}$ is 0. The higher the value of MRR, the better it speaks of potential effort spent in noise. For example, a method with perfect ranking should achieve MRR value of 1; MRR value of 0.5 suggests that average correct answer is found at the second rank.

In order to compare two methods, we define the gain to compare how the method 1 outperforms the method 2. As described in initial study [20], accuracy gain and MRR gain are defined as follows:

$$Gain_{accuracy_m} = \frac{(Accuracy_m(1) - Accuracy_m(2))}{Accuracy_m(2)} \tag{8}$$

$$Gain_{MRR} = \frac{(MRR(1) - MRR(2))}{MRR(2)} \tag{9}$$

where $Accuracy_m(1)$ and $MRR(1)$ evaluates the performance of method 1, and $Accuracy_m(2)$ and $MRR(2)$ evaluates the performance of method 2. If the gain value is above 0, it means method 1 has better accuracy than method 2; otherwise method 2 has better prediction results.

Further, we define the following null hypotheses to assess the statistical validity of results. The alternative hypotheses can be easily derived from the respective null hypotheses.

H-1: There is no SSD between $accuracy_m$, MRR values of TRFPre and cHRev.

H-2: There is no SSD between $accuracy_m$, MRR values of TRFPre and WRC.

H-3: There is no SSD between $accuracy_m$, MRR values of TRFPre and TIE.

H-4: There is no SSD between $accuracy_m$, MRR values of TRFPre and CoreDevRec.

H-5: There is no SSD between $accuracy_m$, MRR values of TRFPre and ACRec.

According to previous work [4], we applied One Way ANOVA test to assess statistically significant difference (SSD) with $\alpha = 0.05$ between accuracy and MRR values of compared approaches. Test purpose is to assess whether the distribution of one of the two samples is stochastically greater than the other.

4.4. RQ1: Performance of TRFPre

In order to answer RQ1, we consult Table 6 and Table 7 which shows accuracies and MRR. In project *cocos2d-x*, TRFPre achieves top-1 and top-2 accuracies of 0.76 and 0.94. TRFPre achieves high accuracies in project *cocos2d-x*. In 14

Table 7: MRR of Approaches cHRev, WRC, TIE, CoreDevRec, ACRec and TRFPre. (Best results in bold.)

Project	**MRR**					
	cH-Rev	WRC	TIE	CoreD-evRec	ACRec	TRF-Rec
rails	0.36	0.43	0.39	0.52	0.48	**0.53**
commcare-hq	0.42	0.46	0.49	0.52	0.48	**0.54**
tgstation	0.31	0.37	0.33	0.51	0.41	**0.54**
symfony	0.62	0.92	0.86	**0.93**	0.82	**0.93**
cocos2d-x	0.72	0.72	0.78	0.82	0.85	**0.87**
core	0.39	0.56	0.44	0.61	0.44	**0.63**
Baystation12	0.42	0.52	0.46	0.58	0.52	**0.61**
joomla-cms	0.29	0.33	0.37	0.51	0.35	**0.54**
app	0.25	0.13	0.24	0.29	0.18	**0.48**
metasploit-framework	0.41	0.47	0.5	0.54	0.51	**0.59**
bootstrap	0.74	0.76	**0.81**	0.78	0.75	**0.81**
dmd	0.46	0.64	0.55	0.68	0.55	**0.7**
cdnjs	0.72	0.59	0.77	0.82	**0.85**	0.84
zendframework	0.64	0.74	0.77	0.76	0.76	**0.78**
angular.js	0.45	0.45	0.51	0.53	**0.55**	**0.55**
cakephp	0.61	0.73	0.72	0.78	0.72	**0.79**
puppet	0.35	0.29	0.41	0.45	0.42	**0.5**
brackets	0.4	0.37	0.44	0.46	0.43	**0.5**
scala	0.51	0.5	0.58	0.62	0.63	**0.67**
ipython	0.54	0.58	0.57	0.65	0.6	**0.68**
sympy	0.47	0.45	0.51	0.51	0.51	**0.56**
node-v0.x-archive	0.49	0.43	0.56	0.61	0.58	**0.64**
wet-boew	0.61	0.69	0.67	0.7	0.68	**0.76**
katello	0.37	0.32	0.37	0.38	0.39	**0.47**
Average	0.48	0.52	0.55	0.61	0.56	**0.65**

projects, TRFPre achieves top-2 accuracies higher than 0.6. On average, TRFPre achieves top-1 accuracy, top-2 accuracy, and MRR of 0.48, 0.67 and 0.65. TRFPre achieves MRR values greater than 0.5 in 91.67% of projects. That is, on average a maximum of 2 predictions need to be examined to get correct integrator.

RQ1: TRFPre makes accurate integrator predictions in terms of accuracies and MRR. Less than 2 predictions are needed to find correct integrator in 91.67% of projects.

4.5. RQ2: Approach Comparison

Table 6 and Table 7 show accuracies and MRR of cHRev [4], WRC [5], TIE [6], CoreDevRec [10], ACRec [8] and our approach TRFPre. In order to compare TRFPre with other approaches, we compute accuracy gains and MRR gains, assess the statistically significant difference between approaches, and describe results in Table 8 and Table 9. Table 10 shows the total model construction time for the 24 projects (in hours) and average prediction time per pull request (in seconds). cHRev, WRC and ACRec do not have model construction phase. TRFPre's total model construction time is 14.3 hours, while its average prediction time per pull request is only 0.07 seconds. In practice, TRFPre can construct models overnight, and the models can be used to predict integrators for many pull requests. Each time a pull request is processed, only a fraction of a second is needed to produce a list of potential integrators.

First, we compare accuracy and MRR values between cHRev and TRFPre. On average across 24 projects, TRFPre outperform cHRev results by 68.2%, 51.4% and 38.7% in terms of top-1 accuracy, top-2 accuracy and MRR. Clearly, TRFPre outperforms cHRev across accuracy and MRR values in all 24 projects. TRFPre records positive gains with statistical significance (with p-values<0.05) in all cases. Therefore, we find support to reject Hypothesis H-1 in favor of TRFPre. As shown in Table 10, TRFPre has longer prediction time than cHRev. cHRev directly computes scores of integrators based on their previous histories, and thus make quicker prediction than TRFPre.

Table 9: MRR Gains and Statistical Results of Approaches cHRev, WRC, TIE, CoreDevRec, ACRec and TRFPre.

Project	**MRR Gain %**				
			TRFPre-		
	cHRev	WRC	TIE	CoreD-evRec	ACRec
rails	47.2 ***	23.3 ***	35.9 ***	1.9	10.4 ***
commcare-hq	28.6 ***	17.4 ***	10.2 ***	3.8 ***	12.5 ***
tgstation	74.2 ***	45.9 ***	63.6 ***	5.9 ***	31.7 ***
symfony	50 ***	1.1 **	8.1 ***	0	13.4 ***
cocos2d-x	20.8 ***	20.8 ***	11.5 ***	6.1 ***	2.4 ***
core	61.5 ***	12.5 ***	43.2 ***	3.3 **	43.2 ***
Baystation12	45.2 ***	17.3 ***	32.6 ***	5.2 ***	17.3 ***
joomla-cms	86.2 ***	63.6 ***	45.9 ***	5.9 ***	54.3 ***
app	92 ***	269.2 ***	100 ***	65.5 ***	166.7 ***
metasploit-framework	43.9 ***	25.5 ***	18 ***	9.3 ***	15.7 ***
bootstrap	9.5 ***	6.6 ***	0	3.8 ***	8 ***
dmd	52.2 ***	9.4 ***	27.3 ***	2.9 *	27.3 ***
cdnjs	16.7 ***	42.4 ***	9.1 ***	2.4 ***	-1.2
zendframework	21.9 ***	5.4 ***	1.3 *	2.6 **	2.6 ***
angular.js	22.2 ***	22.2 ***	7.8 ***	3.8 *	0
cakephp	29.5 ***	8.2 ***	9.7 ***	1.3	9.7 ***
puppet	42.9 ***	72.4 ***	22 ***	11.1 ***	19 ***
brackets	25 ***	35.1 ***	13.6 ***	8.7 ***	16.3 ***
scala	31.4 ***	34 ***	15.5 ***	8.1 ***	6.3 ***
ipython	25.9 ***	17.2 ***	19.3 ***	4.6 ***	13.3 ***
sympy	19.1 ***	24.4 ***	9.8 ***	9.8 ***	9.8 ***
node-v0.x-archive	30.6 ***	48.8 ***	14.3 ***	4.9 *	10.3 ***
wet-boew	24.6 ***	10.1 ***	13.4 ***	8.6 ***	11.8 ***
katello	27 ***	46.9 ***	27 ***	23.7 ***	20.5 ***
Average	38.7	36.7	23.3	8.5	21.7

$***p < 0.001, **\ p < 0.01, *p < 0.05$

Table 10: Total model construction time (in hours) and average prediction time per pull request (in seconds) for cHRev, TIE, CoreDecRec, ACRec, and TRFRec for the 24 projects.

Approach	**Model construction time (in hours)**	**Prediction time (in seconds)**
cHRev	NA	0.04
WRC	NA	0.01
TIE	4.2	0.05
CoreDevRec	120.5	0.2
ACRec	NA	0.04
TRFRec	14.3	0.07

Next, we compare accuracy and MRR values between TRFPre and WRC. In project *tgstation*, TRFPre achieves top-1 accuracy, top-2 accuracy and MRR values of 0.33, 0.54 and 0.54, which outperform WRC results by 50%, 74.2% and 45.9%, respectively. On average across 24 projects, TRFPre outperforms WRC results by 73.9%, 61% and 36.7% in terms of top-1 accuracy, top-2 accuracy and MRR. Clearly, TRFPre outperforms WRC across accuracy and MRR values. Furthermore, p-values are smaller than 0.05 in most of cases, and TRFPre records positive gains with statistical significance. Therefore, we find support to reject Hypothesis H-2 in favor of TRFPre. Since WRC has the simplest prediction algorithm and no model construction phase, WRC has the shorted prediction time.

Thirdly, we compare performance between TRFPre and TIE. On average across 24 projects, TRFPre outperforms TIE results by 49.3%, 30.6% and 23.3% in terms of top-1 accuracy, top-2 accuracy and MRR. TRFPre achieves statistically significant higher accuracy and MRR values than TIE in most of cases. Therefore, we find support to reject Hypothesis H-3 in favor of TRFPre. The model construction time of pull requests is 4.2 hours and 14.3 hours for TIE and TRFPre, respectively. TIE costs shorter model construction time than TRFPre. This because TIE has 1 classifier, while TRFPre has 2 classifiers in model construction.

Fourthly, we make comparison between CoreDevRec and TRFPre. In project *rails*, TRFPre has lower top-1 accuracy than CoreDevRec, but TRFPre has higher top-2 accuracy and MRR than CoreDevRec. On average across 24 projects, TRFPre improves CoreDevRec by 14.3%, 10.5% and 8.5% in terms of top-1 accuracy, top-2 accuracy and MRR. The statistical testing results show that most of p-values are smaller than 0.05, and thus we find support to reject Hypothesis H-4. TRFPre achieves higher accuracy and MRR values, because TRFPre considers time-decaying relationships and time-decaying file similarities. CoreDevRec independently analyzes file paths, social connections and integrators' activeness. However, CoreDevRec does not combine activeness with other features. Pull requests created at different time have the same weights in analysis of file paths and social connections.

In Table 10, CoreDevRec costs 120.5 hours for model construction, which is much larger than TRFPre. This is because CoreDevRec uses support vector machines to analyze many elements. NV is the number of substrings in file path vocabulary. In project cocos2d-x, NV is as large as 18,018. In the majority of projects, NV is more than 4,000. The number of integrators in a project is denoted as NC. For CoreDevRec, each pull request vector has $NV + 10 \times NC$ elements. As described in subsection 3.5, each pull request vector has $2 \times NC$ elements for TRFPre, which is much smaller than CoreDevRec. Therefore, TRFPre has shorter model construction time than CoreDevRec.

Finally, we compare performance between TRFPre and ACRec. On average across 24 projects, TRFPre outperforms ACRec results by 46.4%, 29.6% and 21.7% in terms of top-1 accuracy, top-2 accuracy and MRR. The statistical testing results show that most of p-values are smaller than 0.05, and thus we find support to reject Hypothesis H-5. ACRec costs shorter prediction time than TRFPre, because it simply computes developers' activeness to make prediction.

RQ2: TRFPre achieves statistically significant higher accuracies and MRRs than cHRev, WRC, TIE, CoreDevRec and ACRec.

4.6. RQ3: Configuration Setting

In order to answer RQ3, we build another approach RFRec which uses non time-decaying relationships and file similarities, and treats all pull requests with the same weigh 1. More specifically, non time-decaying relationship can be measured by $|OSet_{new} \bigcap PSet_i|$. Non time-decaying file similarity can be measured by $\sum_{P_j \in PSet_i} prDistance(P_{new}, P_j)$. Based on these features, RFRec also uses random forest classifier and naive bayes classifier to compute likelihoods, and combine classification results to make prediction.

Table 11: Top-m Accuracies (m=1,2) and MRR of Approaches RFRec and TRFPre. (Best results in bold.)

Project	**Top-1 Accuracy**		**Top-2 Accuracy**		**MRR**	
	RFRev	TRFPre	RFRev	TRFPre	RFRev	TRFPre
rails	0.22	**0.36**	0.41	**0.53**	0.42	**0.53**
commcare-hq	0.24	**0.36**	0.39	**0.54**	0.43	**0.54**
tgstation	0.27	**0.33**	0.47	**0.54**	0.49	**0.54**
symfony	0.85	**0.89**	0.91	**0.94**	0.9	**0.93**
cocos2d-x	0.71	**0.76**	0.92	**0.94**	0.83	**0.87**
core	0.35	**0.47**	0.5	**0.65**	0.53	**0.63**
Baystation12	0.3	**0.38**	0.55	**0.65**	0.54	**0.61**
joomla-cms	0.22	**0.33**	0.44	**0.54**	0.45	**0.54**
app	0.27	**0.32**	0.42	**0.48**	0.43	**0.48**
metasploit -framework	0.31	**0.41**	0.49	**0.6**	0.51	**0.59**
bootstrap	0.64	**0.67**	0.83	**0.9**	0.78	**0.81**
dmd	0.44	**0.53**	0.59	**0.72**	0.61	**0.7**
cdnjs	0.7	**0.74**	**0.9**	**0.9**	0.83	**0.84**
zendframework	0.58	**0.66**	0.75	**0.83**	0.73	**0.78**
angular.js	0.33	**0.35**	0.48	**0.55**	0.51	**0.55**
cakephp	0.56	**0.65**	0.72	**0.85**	0.72	**0.79**
puppet	0.28	**0.33**	0.44	**0.5**	0.46	**0.5**
brackets	0.22	**0.32**	0.38	**0.49**	0.42	**0.5**
scala	0.39	**0.47**	0.61	**0.71**	0.6	**0.67**
ipython	0.45	**0.49**	0.67	**0.75**	0.65	**0.68**
sympy	0.27	**0.37**	0.5	**0.57**	0.49	**0.56**
node-v0.x-archive	0.43	**0.46**	0.63	**0.66**	0.62	**0.64**
wet-boew	0.49	**0.59**	0.78	**0.86**	0.69	**0.76**
katello	0.25	**0.29**	0.43	**0.45**	0.45	**0.47**

Table 12: Gains and Statistical Results of Approaches RFRec and TRFPre.

Project	**Top-1 Accuracy Gain %**	**Top-2 Accuracy Gain %**	**MRR Gain %**
rails	63.6 ***	29.3 ***	26.2 ***
commcare-hq	50 ***	38.5 ***	25.6 ***
tgstation	22.2 ***	14.9 ***	10.2 ***
symfony	4.7 ***	3.3 ***	3.3 ***
cocos2d-x	7 ***	2.2 ***	4.8 ***
core	34.3 ***	30 ***	18.9 ***
Baystation12	26.7 ***	18.2 ***	13 ***
joomla-cms	50 ***	22.7 ***	20 ***
app	18.5 ***	14.3 ***	11.6 ***
metasploit-framework	32.3 ***	22.4 ***	15.7 ***
bootstrap	4.7 *	8.4 ***	3.8 ***
dmd	20.5 ***	22 ***	14.8 ***
cdnjs	5.7 ***	0	1.2 ***
zendframework	13.8 ***	10.7 ***	6.8 ***
angular.js	6.1 **	14.6 ***	7.8 ***
cakephp	16.1 ***	18.1 ***	9.7 ***
puppet	17.9 ***	13.6 ***	8.7 ***
brackets	45.5 ***	28.9 ***	19 ***
scala	20.5 ***	16.4 ***	11.7 ***
ipython	8.9 ***	11.9 ***	4.6 ***
sympy	37 ***	14 ***	14.3 ***
node-v0.x-archive	7 *	4.8 *	3.2
wet-boew	20.4 ***	10.3 ***	10.1 ***
katello	16 ***	4.7 *	4.4 *
Average	22.9	15.6	11.2

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

Table 13: Performance with different γ

γ	Top-1 Accuracy	Top-2 Accuracy	MRR
0	0.46	0.64	0.62
0.1	0.46	0.65	0.63
0.2	0.47	0.65	0.63
0.3	0.47	0.66	0.64
0.4	0.48	0.67	0.64
0.5	0.48	0.67	0.64
0.6	0.48	0.67	0.64
0.7	0.48	0.67	0.65
0.8	0.48	0.67	0.64
0.9	0.47	0.67	0.64
1	0.46	0.66	0.63

Table 14: Performance with different λ (days)

λ	Top-1 Accuracy	Top-2 Accuracy	MRR
5	0.43	0.61	0.6
10	0.45	0.63	0.62
30	0.47	0.65	0.63
60	0.47	0.66	0.64
120	0.47	0.67	0.64
no time limit	0.48	0.67	0.65

Table 11 compare accuracy and MRR values of RFRec and TRFPre. We compute accuracy gains and MRR gains, assess the statistically significant difference between approaches, and describe results in Table 12. On average across 24 projects, TRFPre improves RFRec by 22.9%, 15.6% and 11.2% in terms of top-1 accuracy, top-2 accuracy and MRR. Furthermore, p-values are smaller than 0.05 in almost all cases, and TRFPre achieves positive gains with statistical significance. The only difference between RFRec and TRFPre is time-decaying function. Results show that time-decaying function is effective to improve performance of integrator prediction.

The parameter γ is used to assign weights of prediction likelihoods of naive bayes classifier and random forest classifier. We increases parameter γ from 0 to 1 with an interval of 0.1, and evaluate performance of TRFPre. We compute average values of 24 projects, and describe results in Table 13. Results show that TRFPre achieves the highest values of accuracies and MR-R when parameter γ is set as 0.7. Therefore, we set parameter γ as 0.7 by default. When parameter γ is set as 0, TRFPre only uses naive bayes classifier to predict integrators. When parameter γ is set as 1, TRFPre only uses random forest classifier to predict integrators. The best performance is achieved when parameter γ is set as 0.7. Results show that the combination of naive bayes classifier and random forest classifier achieves better performance than a single classifier.

By using two machine learning techniques, TRFPre can increase top-1 accuracy from 0.46 to 0.48 (i.e., 4.3% improvement). This improvement is achieved at the cost of increasing training and prediction time. Training can be done once and the resultant trained model can be used to predict integrators for many pull requests. For prediction time, by using two machine learning techniques, the average increase in prediction time per pull request is 0.01 seconds (as compared to using Random Forest alone) and 0.02 seconds (as compared to using Naive Bayes alone). Since users would not be able to differentiate a 0.01 or 0.02 seconds prediction time difference, the results show that a negligible increase in prediction time per pull request can yield a small improvement in accuracy.

In Equations1 and 4, $PSet_i$ includes pull requests on which integrator D_i has made decisions within λ days before new pull request P_{new}. By default, we set the temporal window length λ as infinite (i.e., no time limit), and consider *all* pull requests on which integrators have made decision on. Here, we investigate the effect of the temporal window length λ on the performance of TRFPre. We compute the approach performance with different λ values, including: 5 days, 10 days, 30 days, 60 days, 120 days and no time limit. We compute average values of the 24 projects, and present the results in Table 14.

Results show that as λ increases, top-1 and top-2 accuracies and MRR also increase. However, the increase in these evaluation metrics is getting lesser and lesser for larger λ. TRFPre achieves the highest values of accuracies and MRR when parameter λ is set to no time limit. The results suggest that old data are also useful albeit they are less important than more recent data. Also, since the performance at λ = *no time limit* is the best, we use it as the default setting of TRFPre.

RQ3: The time-decaying function is effective for integrator prediction. TRFPre achieves the best accuracy and MRR values when parameter γ is set as 0.7 and parameter λ is set as no time limit.

5. Discussion

In this section, we first discuss why we choose actual integrators as ground truth. Then we explore feature importance in integrator prediction.

5.1. Ground truth

In this paper, we consider ground truth as integrators who actually evaluate pull requests. We do not know whether historical integrators are the best or appropriate for making decisions on pull requests. Contributors submit pull requests, which are evaluated by integrators. We send a survey to contributors, and explore their attitudes towards integrators. More specifically, we design a survey to include 2 questions.

1. *In your experience, how often is an actual integrator of a pull request the BEST person for reviewing code and deciding its acceptance/rejection?*

2. *In your experience, how often is an actual integrator of a pull request an INAPPROPRIATE person for reviewing code and deciding its acceptance/rejection?*

We provide nine choices for above questions including: 'Never', 'Almost Never', 'Very Rarely', 'Rarely', 'Sometimes', 'Often', 'Very Often', 'Almost Always', 'Always'.

We randomly selected 300 contributors who submitted at least 5 pull requests in a project in May 2018. We sent them emails, and asked the above questions. We received responses from 24 developers. Then, we randomly selected another 300 contributors who submitted at least 5 pull requests in a project,

Table 15: Distribution of the number of integrators

# Integrators	# Projects
[10, 20)	9
[20, 30)	5
[30, 40)	4
[40, 50)	4
[50, 100)	2

Table 16: How often is an actual integrator of a pull request the BEST person for reviewing code and deciding its acceptance/rejection?

Choice	Respondents
Never	0 / 0 %
Almost Never	0 / 0 %
Very Rarely	1 / 2.04 %
Rarely	1 / 2.04 %
Sometimes	4 / 8.16 %
Often	11 / 22.45 %
Very Often	15 / 30.61 %
Almost Always	12 / 24.5 %
Always	5 / 10.2 %

sent them emails, and received responses from 25 developers in November 2018. In total, we obtained responses from 49 developers. These developers are from projects which have at least 10 integrators. These projects include *rails*, *symfony*, *angular.js* and so on. Table 15 shows the distribution of the number of integrators. 10 projects have at least 30 integrators, and the number of integrators in the other 14 projects is between 10 and 30.

We ask developers how often an actual integrator of a pull request is the BEST person for reviewing code and deciding its acceptance/rejection. We plot their responses in Table 16. From the table, we can note that 87.76% of our respondents agree that it is often/very often/almost always/always the case that actual integrators are the best ones.

Table 17 shows developers' attitude about how often an actual integrator of a pull request is the INAPPROPRIATE person for reviewing and deciding its acceptance/rejection. From the table, we can note that close to 77.55% of our respondents agree that it is rarely/very rarely/almost never/never the case that actual integrators are inappropriate ones.

Table 17: How often is an actual integrator of a pull request the INAPPROPRIATE person for reviewing code and deciding its acceptance/rejection?

Choice	Respondents
Never	3 / 6.12 %
Almost Never	9 / 18.37 %
Very Rarely	16 / 32.65 %
Rarely	10 / 20.41 %
Sometimes	8 / 16.33 %
Often	2 / 4.08 %
Very Often	1 / 2.04 %
Almost Always	0 / 0 %
Always	0 / 0 %

Table 18: Logistic regression analysis results

Feature	Odds ratio
(Intercept)	2.99E-03 ***
decayRelationship	1.2 ***
decayFileSimilarity	1.16 ***
recentCommit	1.12 ***
recentComment	1.14 ***
recentPulls	1.13 ***
followerRelation	1.03 ***
followingRelation	1.06 ***
evaluateTime	0.97 ***
evaluatePulls	0.88 ***
latestTime	0.84 ***

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

The above survey results indicate that although our ground truth data may not be perfect they are of reasonably high-quality.

5.2. Feature Importance

This subsection presents a preliminary study to explore the relative importance of features to the selection of integrators. We investigate the importance of a total of 10 features; two of them are proposed in this work while the other eight are suggested in prior studies [8, 10]. They are listed below:

1. *decayRelationship* is proposed in this work and defined in Equation 1.
2. *decayFileSimilarity* is also proposed in this work and defined in Equation 4.
3. *recentCommit* is the number of commits submitted by an integrator in 30 days.
4. *recentComment* is the number of comments submitted by an integrator in 30 days.
5. *recentPulls* is the number of pull requests evaluated by an integrator in 30 days.
6. *followerRelation* is a dichotomous variable indicating whether the contributor is the follower of the integrator or not.
7. *followingRelation* is a dichotomous variable indicating whether or not the integrator follows the contributor.
8. *evaluateTime* is computed as the average interval time between the pull request submission and the integrator evaluation in recent 30 days.
9. *evaluatePulls* is the total number of pull requests evaluated by the integrator before.
10. *latestTime* is computed as the interval time between close time of the latest evaluated pull request by an integrator and the new submitted pull request.

Logistic regression is used to model binary outcome variables, in which the log odds of the outcomes are modeled as a linear combination of the predictor variables. Following a previous work [21], we use a logistic regression model, and analyze the correlations between various features extracted from pull-request-integrator pairs and binary outcomes each indicating whether a pull request is evaluated by an integrator or not. In order to ensure normality [21], every continuous variable in the model is log transformed and then centered such that the mean of each measure is 0 and standard deviation is 1.

Table 18 summarizes logistic regression analysis results. We compute the correlation between a feature and the probability of evaluating a pull request by an integrator in terms of odds ratio. For example, an odds ratio of 0.6 suggests that as the value of a feature increases by one unit, the odds of evaluating the pull request by an integrator decreases by 40%; an odds ratio of 1.2 suggests that as the value of a feature increases by one unit, the odds of evaluating the pull request by an integrator increases by 20%.

As shown in Table 18, p-values for all features are smaller than 0.001, and there is a statistically significant correlation between features and the likelihood whether integrators evaluate pull requests. The odds ratio of feature *decayRelationship* is 1.2. The feature *decayRelationship* is positively associated with the likelihood of evaluating pull requests, and a unit increase of it increases the odds of evaluating pull requests by 20%. We find that features *decayRelationship* and *decayFileSimilarity* that are proposed in this work have the highest values of positive impact on integrators' likelihoods of evaluating pull requests.

6. Threats to Validity

Threats to internal validity relate to experimenter bias and errors. First, we use decision histories to build relationships between integrators and contributors. There may be some other communication channels that developers use, such as mailing lists, instant messages or face-to-face discussion, and this work ignores them. Future work may consider collecting data from additional communication channels, and analyze its impact on integrator prediction. Second, we identify developers by their usernames in GitHub. A developer may use multiple usernames, which is not considered in this work.

Threats to external validity relate to generalizability of our study. Firstly, our experimental results are limited to 24 popular projects. We find that TRFPre achieves higher accuracy and MRR values than cHRev, WRC, TIE, CoreDevRec and ACRec, which are based on 24 projects in our datasets. We cannot claim that the same results would be achieved in other projects. Our future work will focus on evaluation in other projects to better generalize results of our method. We will conduct broader experiments to validate whether TRFPre performs well in integrator prediction. Secondly, our empirical findings are based on open source software projects in GitHub, and it is unknown whether our results can be generalized to other OSS platforms. In the future, we plan to study a similar set of research questions in other platforms, and compare their results with our findings in GitHub.

Threats to construct validity refer to the degree to which the construct being studied is affected by experiment settings. First, we use accuracy and MRR, which are also used by previous works to evaluate effectiveness of reviewer prediction approaches [6, 18] and various automated software engineering techniques [22, 23, 24]. Therefore, we believe there is little threat to construct validity. Second, we have not studied all potential factors affecting integrator prediction. In future work, we plan to extract more factors and perform an in-depth and comprehensive explanatory study to investigate which of these factors are more important for selection of suitable integrators.

7. Related work

Related work to this study could be divided into three main categories, including reviewer recommendation, developer recommendation and contribution integration.

Reviewer recommendation. There have been a number of studies on reviewer recommendation.

Initial studies [5, 6, 10, 18, 19, 25, 26] analyze different code review activities and design reviewer recommendation approaches. In order to recommend reviewers, approach WRC analyzed past reviews with the same files [5]. Thongtanunam et al. designed a file location-based code-reviewer recommendation approach REVFINDER [18]. Xia et al. proposed a hybrid and incremental approach TIE [6], and achieved better performance than REVFINDER [18]. Since TIE [6] outperforms REVFINDER [18], we choose TIE as a baseline for comparison, rather than REVFINDER [18]. Jiang et al. used support vector machines to analyze integrators' previous decisions, and designed an approach CoreDevRec to recommend integrators [10]. Different from these approaches, we propose integrator prediction approach TRFPre based on time-decaying relationships and time-decaying file similarities. Experiment results show that TRFPre achieves higher accuracies and MRRs than cHRev, WRC, TIE and CoreDevRec.

Previous works [4, 7, 8, 27] designed approaches to recommend commenters who provide comments and discuss pull requests. Zanjani et al. analyzed review comments and proposed a reviewer expertise approach [4]. Yu et al. built comment networks to predict appropriate commenters of incoming pull requests in GitHub [7, 27]. More recently, Jiang et al. proposed the activeness based approach ACRec to recommend commenters [8]. Since ACRec [8] has been demonstrated to outperform Yu et al's approaches, in this work we compare our work only with cHRev and ACRec. Our experiments show that our approach outperforms cHRev and ACRec. This is the case since cHRev and ACRec along with some other existing commenter recommendation methods only analyze previous review comments, and ignore pull requests without comments. However, these pull requests without comments still have integrators and provide valuable historical information for integrator prediction. As shown in Table 2, more than 20% of pull requests integrators do not provide any comments in 18 (75%) projects.

Developer Recommendation. Recommendation systems specific to software engineering (RSSE) help developers in a wide range of activities. Finding developers is an important need in recommendation systems specific to software engineering. Some previous studies designed approaches to assign bug reports or change requests [20, 22, 23, 24, 28, 29, 30, 31, 32, 33, 34]. Anvik et al. applied a machine learning algorithm and suggested a small number of developers to resolve bug reports [22]. Jeong et al. used bug tossing history to assign developers for bug reports [30]. Matter et al. used a text-based method to identify expertise of developers for bug reports [23]. Hossen et al. considered source code authors, maintainers, and change proneness to triage change requests [28]. V¢squez et al. utilized source code authorship for assigning expert developers to change requests [24].

Our approach addresses a different problem (integrator prediction) than those considered in the above mentioned past studies (bug fixer recommendation or change request handler recommendation).

Contribution integration. In pull-based development model, integrators have crucial role of managing and integrating contributions. Some works studied contribution integration in GitHub [1, 15, 35, 36]. Gousios et al. made an exploratory qualitative study to understand integrators' work practices and challenges in GitHub [1]. Gousios et al. took an further step, and studied contributors' work practices and challenges in contribution integration [35]. Vasilescu et al. studied quality and productivity outcomes relating to continuous integration in GitHub [15]. Hilton et al. studied the usage of continuous integration systems in contribution integration [36].

Different from these works, we solve a different problem and design an automatic approach to predict integrators for incoming pull requests.

8. Conclusion

In this paper, we propose an approach TRFPre to predict integrators. TRFPre considers time-decaying relationships and time-decaying file similarities. Based on these attributes, TRFPre uses random forest classifier and naive bayes classifier to compute composite probabilities and predict integrators. We evaluate effectiveness of TRFPre on 24 projects containing 138,373 pull requests. We compare it to the state of art reviewer prediction approaches. The experimental results show that on average across 24 projects, TRFPre improves cHRev, WRC, TIE, CoreDevRec and ACRec by 68.2%, 73.9%, 49.3%, 14.3% and 46.4% in terms of top-1 accuracy. TRFPre achieves better prediction performance than cHRev, WRC, TIE, CoreDevRec and ACRec. When TRFPre is used to predict integrators, less than 2 predictions are needed to find correct integrator in 91.67% of projects. Therefore, we believe that TRFPre is useful to find appropriate integrators and improve code review process.

.

9. Acknowledgment

This work is supported by the National Key Research and Development Program of China No. 2018YFB1004202, the National Natural Science Foundation of China under Grant No. 61732019, and the State Key Laboratory of Software Development Environment under Grant No.SKLSDE-2018ZX-12.

[1] G. Gousios, A. Zaidman, M.-A. Storey, A. van Deursen, Work practices and challenges in pull-based development: The integrators perspective, in: Proc. the 37th ICSE, Florence, Italy, 2015, pp. 1–11.

[2] J. Tsay, L. Dabbish, J. Herbsleb, Let's talk about it: Evaluating contributions through discussion in github, in: Proc. of FSE, Hong Kong, China, 2014, pp. 144–154.

[3] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, in: Proc. of MSR, Buenos Aires, Argentina, 2017, pp. 215–226.

[4] M. B. Zanjani, H. Kagdi, C. Bird, Automatically recommending peer reviewers in modern code review, IEEE Transactions on Software Engineering 42 (6) (2016) 530–543.

[5] C. Hannebauer, M. Patalas, S. Stnkel, Automatically recommending code reviewers based on their expertise: An empirical comparison, in: Proc. of ASE, Singapore, Singapore, 2016, pp. 99–110.

[6] X. Xia, D. Lo, X. Wang, X. Yang, Who should review this change? putting text and file location analyses together for more accurate recommendations, in: Proc. of ICSME, Bremen, Germany, 2015.

[7] Y. Yu, H. Wang, G. Yin, T. Wang, Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?, Information and Software Technology 74 (2016) 204–218.

[8] J. Jiang, Y. Yang, J. He, X. Blanc, L. Zhang, Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development, Information and Software Technology 84 (2017) 48–62.

[9] G. Robles, J. M. Gonzalez-Barahona, Contributor turnover in libre software projects, in: Open Source Systems, 2006, pp. 273–286.

[10] J. Jiang, J.-H. He, X.-Y. Chen, Coredevrec: Automatic core member recommendation for contribution evaluation, Journal of Computer Science and Technology 30 (5) (2015) 998–1016.

[11] W. Li, Y. Xia, M. Zhou, X. Sun, Q. Zhu, Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds, IEEE Access (2018) 1–16.

[12] Y. Yu, G. Yin, T. Wang, C. Yang, H. Wang, Determinants of pull-based development in the context of continuous integration, Science China Information Sciences 59 (8) (2016) 1–14.

[13] V. J. Hellendoorn, P. T. Devanbu, A. Bacchelli, Will they like this? evaluating code contributions with language models, in: Proc. of the 12nd MSR, Florence, Italy, 2015.

[14] A. Bacchelli, C. Bird, Expectations, outcomes, and challenges of modern code review, in: Proc. of ICSE, San Francisco, USA, 2013.

[15] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in github, in: Proc. of FSE, Bergamo, Italy, 2015.

[16] G. Gousios, M. Pinzger, A. van Deursen, An exploratory study of the pull-based software development model, in: Proc. the 36th ICSE, Hyderabad, India, 2014, pp. 345–355.

[17] W. Zhang, L. Nie, H. Jiang, Z. Chen, J. Liu, Developer social networks in software engineering: construction, analysis, and applications, SCIENCE CHINA Information Sciences 57 (12) (2014) 1–23.

[18] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, K. ichi Matsumoto, Who should review my code? a file location-based code-reviewer recommendation approach for modern code review, in: Proc. the 22nd SANER, Montreal, Canada, 2015, pp. 141–150.

[19] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in: Proc. of ICSE, San Francisco, USA, 2013, pp. 931–940.

[20] X. Xia, D. Lo, X. Wang, B. Zhou, Accurate developer recommendation for bug resolution, in: Proc. the 20th WCRE, Koblenz, Germany, 2013, pp. 72–81.

[21] J. Tsay, L. Dabbish, J. Herbsleb, Influence of social and technical factors for evaluating contribution in github, in: Proc. the 36th ICSE, Hyderabad, India, 2014, pp. 356–366.

[22] J. Anvik, L. Hiew, G. C. Murphy, Who should fix this bug?, in: Proc. the 28th ICSE, Shanghai, China, 2006, pp. 361–370.

[23] D. Matter, A. Kuhn, O. Nierstrasz, Assigning bug reports using a vocabulary-based expertise model of developers, in: Proc. the 6th MSR, Vancouver, Canada, 2009, pp. 131–140.

[24] M. Linares-Vasquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, D. Poshyvanyk, Triaging incoming change requests: Bug or commit history, or code authorship?, in: Proc. the 28th ICSM, Riva del Garda, Italy, 2012, pp. 451–460.

[25] A. Ouni, R. G. Kula, K. Inoue, Search-based peer reviewers recommendation in modern code review, in: Proc. of ICSME, Raleigh, USA, 2016, pp. 367–377.

[26] M. M. Rahman, C. K. Roy, J. A. Collins, Correct: Code reviewer recommendation in github based on cross-project and technology experience, in: Proc. of ICSE Companion, Austin, USA, 2016, pp. 222–231.

[27] Y. Yu, H. Wang, G. Yin, C. Ling, Reviewer recommender of pull-requests in github, in: Proc. the 30th ICSME, Victoria, Canada, 2014, pp. 609–612.

[28] M. K. Hossen, H. Kagdi, D. Poshyvanyk, Amalgamating source code authors, maintainers, and change proneness to triage change requests, in: Proc. the 22nd ICPC, Hyderabad, India, 2014, pp. 1–12.

[29] H. Kagdi, M. Gethers, D. Poshyvanyk, M. Hammad, Assigning change requests to software developers, JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS 24 (2012) 3–33.

[30] G. Jeong, S. Kim, T. Zimmermann, Improving bug triage with bug tossing graphs, in: Proc. the 17th FSE, Amsterdam, The Netherlands, 2009, pp. 111–120.

[31] H. Hu, H. Zhang, J. Xuan, W. Sun, Effective bug triage based on historical bug-fix information, in: Proc. the 25th ISSRE, Naples, Italy, 2014, pp. 122–132.

[32] W. Wu, Q. W. Wen Zhang, Ye Yang, Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking, in: Proc. the 18th APSEC, Phan Thiet, Vietnam, 2011, pp. 389–396.

[33] D. Cubranic, G. C. Murphy, Automatic bug triage using text categorization, in: Proc. the 16th SEKE, 2004, pp. 1–6.

[34] H. Liu, Z. Ma, W. Shao, Z. Niu, Schedule of bad smell detection and resolution: A new way to save effort, IEEE Transactions on Software Engineering 38 (1) (2012) 220–235.

[35] G. Gousios, M.-A. Storey, A. Bacchelli, Work practices and challenges in pull-based development: The contributors perspective, in: Proc. of ICSE, Austin, TX, USA, 2016.

[36] M. Hilton, T. Tunnell, K. Huang, D. Marinov, D. Dig, Usage, costs, and benefits of continuous integration in open-source projects, in: Proc. of ASE, Signore, Signore, 2016, pp. 426–437.

Table 4: Methods in previous works

Method	**Supervised learning technique**	**Proposed metrics**	**Pros**	**Cons**
cHRev [4]	None	cHRev analyzes review comments and computes reviewers' frequency, workdays, and recency considering specific files.	cHRev does not have time overhead of model construction phase. cHRev considers activeness of reviewers.	cHRev analyzes review comments while many pull requests in GitHub are not commented.
WRC [5]	None	WRC computes weighted review count which specifies the review experience a developer has with a specific file at the time a specific review is made.	WRC does not have time overhead of model construction phase. WRC considers activeness of reviewers based on past reviews.	WRC does not consider file similarities.
TIE [6]	Naive Bayes classifier	TIE proposes a hybrid and incremental approach which analyzes textual content and file paths.	TIE combines text similarity and file path similarity together to make recommendation.	TIE considers previous reviews uploaded within 100 days before new review as equally important.
CoreDevRec [10]	SVM	CoreDevRec analyzes file paths, social interactions and activeness to recommend integrators.	CoreDevRec fully considers file similarity, relation similarity and activeness.	CoreDevRec uses support vector machines to analyze many substrings in file paths, and thus needs long model construction phase time. Additionally, the large number of sub-strings may introduce noise affecting its accuracy.
ACRec [8]	None	ACRec computes recency based on comments.	ACRec does not have time overhead of model construction phase. ACRec considers commenters' activeness	ACRec does not consider uncommented pull requests.

Table 6: Top-m Accuracies (m=1,2) of Approaches cHRev, WRC, TIE, CoreDevRec, ACRec and TRFPre. (Best results in bold.)

Project	Top-1 Accuracy						Top-2 Accuracy					
	cH-Rev	WRC	TIE	CoreD-evRec	ACRec	TRF-Rec	cH-Rev	WRC	TIE	CoreD-evRec	ACRec	TRF-Rec
rails	0.2	0.3	0.16	**0.37**	0.31	0.36	0.31	0.4	0.4	0.51	0.49	**0.53**
commcare-hq	0.25	0.31	0.31	0.35	0.31	**0.36**	0.39	0.42	0.48	0.5	0.46	**0.54**
tgstation	0.14	0.22	0.14	0.29	0.22	**0.33**	0.25	0.31	0.28	0.53	0.38	**0.54**
symfony	0.42	**0.89**	0.78	**0.89**	0.69	**0.89**	0.65	0.9	0.9	0.93	0.9	**0.94**
cocos2d-x	0.56	0.55	0.61	0.68	0.73	**0.76**	0.78	0.78	0.89	0.92	**0.94**	**0.94**
core	0.21	0.42	0.23	0.43	0.21	**0.47**	0.34	0.46	0.39	**0.66**	0.41	0.65
Baystation12	0.2	0.32	0.21	0.36	0.29	**0.38**	0.37	0.5	0.45	0.61	0.52	**0.65**
joomla-cms	0.13	0.14	0.18	0.3	0.16	**0.33**	0.23	0.27	0.31	0.5	0.29	**0.54**
app	0.15	0.04	0.13	0.16	0.07	**0.32**	0.23	0.07	0.2	0.25	0.13	**0.48**
metasploit-framework	0.24	0.28	0.3	0.35	0.3	**0.41**	0.38	0.45	0.52	0.55	0.52	**0.6**
bootstrap	0.59	0.6	**0.68**	0.61	0.55	0.67	0.8	0.85	0.89	**0.9**	0.89	**0.9**
dmd	0.24	0.49	0.32	0.52	0.33	**0.53**	0.43	0.62	0.54	0.69	0.54	**0.72**
cdnjs	0.54	0.4	0.63	0.71	**0.75**	0.74	0.8	0.57	0.81	0.86	0.89	**0.9**
zendframework	0.49	0.62	0.65	0.63	0.63	**0.66**	0.66	0.74	0.79	0.8	0.79	**0.83**
angular.js	0.28	0.28	0.31	0.32	**0.35**	**0.35**	0.43	0.41	0.5	0.54	0.54	**0.55**
cakephp	0.4	0.54	0.53	0.64	0.53	**0.65**	0.65	**0.85**	0.79	**0.85**	0.8	**0.85**
puppet	0.21	0.13	0.24	0.28	0.24	**0.33**	0.32	0.23	0.38	0.42	0.4	**0.5**
brackets	0.23	0.2	0.25	0.28	0.23	**0.32**	0.37	0.32	0.41	0.43	0.38	**0.49**
scala	0.3	0.27	0.35	0.42	0.42	**0.47**	0.51	0.51	0.6	0.65	0.66	**0.71**
ipython	0.3	0.36	0.33	0.46	0.36	**0.49**	0.55	0.61	0.61	0.69	0.64	**0.75**
sympy	0.3	0.24	0.29	0.31	0.29	**0.37**	0.44	0.51	0.5	0.53	0.5	**0.57**
node-v0.x-archive	0.32	0.26	0.38	0.43	0.42	**0.46**	0.49	0.4	0.58	0.64	0.62	**0.66**
wet-boew	0.39	0.5	0.45	0.51	0.46	**0.59**	0.65	0.77	0.76	0.8	0.77	**0.86**
katello	0.19	0.14	0.17	0.19	0.2	**0.29**	0.33	0.27	0.31	0.33	0.36	**0.45**
Average	0.3	0.35	0.36	0.44	0.38	**0.48**	0.47	0.51	0.55	0.63	0.58	**0.67**

Table 8: Top-m Accuracy (m=1,2) Gains and Statistical Results of Approaches cHRev, WRC, TIE, CoreDevRec, ACRec and TRFPre.

Project	Top-1 Accuracy Gain %					Top-2 Accuracy Gain %				
	cHRev	WRC	TRFPre-TIE	CoreD-evRec	ACRec	cHRev	WRC	TRFPre-TIE	CoreD-evRec	ACRec
rails	80 ***	20 ***	125 ***	-2.7	16.1 ***	71 ***	32.5 ***	32.5 ***	3.9 **	8.2 ***
commcare-hq	44 ***	16.1 ***	16.1 ***	2.9	16.1 ***	38.5 ***	28.6 ***	12.5 ***	8 ***	17.4 ***
tgstation	135.7 ***	50 ***	135.7 ***	13.8 ***	50 ***	116 ***	74.2 ***	92.9 ***	1.9	42.1 ***
symfony	109.5 ***	0	12.8 ***	0	29 ***	44.6 ***	4.4 ***	4.4 ***	1.1	4.4 ***
cocos2d-x	35.7 ***	38.2 ***	24.6 ***	11.8 ***	4.1 ***	20.5 ***	20.5 ***	5.6 ***	2.2 ***	0
core	123.8 ***	11.9 ***	104.3 ***	9.3 ***	123.8 ***	91.2 ***	41.3 ***	66.7 ***	-1.5	58.5 ***
Baystation12	90 ***	18.8 ***	81 ***	5.6 ***	31 ***	75.7 ***	30 ***	44.4 ***	6.6 ***	25 ***
joomla-cms	153.8 ***	135.7 ***	83.3 ***	10 ***	106.3 ***	134.8 ***	100 ***	74.2 ***	8 ***	86.2 ***
app	113.3 ***	700 ***	146.2 ***	100 ***	357.1 ***	108.7 ***	585.7 ***	140 ***	92 ***	269.2 ***
metasploit-framework	70.8 ***	46.4 ***	36.7 ***	17.1 ***	36.7 ***	57.9 ***	33.3 ***	15.4 ***	9.1 ***	15.4 ***
bootstrap	13.6 ***	11.7 ***	-1.5	9.8 ***	21.8 ***	12.5 ***	5.9 ***	1.1	0	1.1
dmd	120.8 ***	8.2 ***	65.6 ***	1.9	60.6 ***	67.4 ***	16.1 ***	33.3 ***	4.3 **	33.3 ***
cdnjs	37 ***	85 ***	17.5 ***	4.2 **	-1.3	12.5 ***	57.9 ***	11.1 ***	4.7 ***	1.1
zendframework	34.7 ***	6.5 ***	1.5	4.8 *	4.8 **	25.8 ***	12.2 ***	5.1 ***	3.7 ***	5.1 ***
angular.js	25 ***	25 ***	12.9 ***	9.4 **	0	27.9 ***	34.1 ***	10 ***	1.9	1.9
cakephp	62.5 ***	20.4 ***	22.6 ***	1.6	22.6 ***	30.8 ***	0	7.6 ***	0	6.2 ***
puppet	57.1 ***	153.8 ***	37.5 ***	17.9 ***	37.5 ***	56.3 ***	117.4 ***	31.6 ***	19 ***	25 ***
brackets	39.1 ***	60 ***	28 ***	14.3 ***	39.1 ***	32.4 ***	53.1 ***	19.5 ***	14 ***	28.9 ***
scala	56.7 ***	74.1 ***	34.3 ***	11.9 ***	11.9 ***	39.2 ***	39.2 ***	18.3 ***	9.2 ***	7.6 ***
ipython	63.3 ***	36.1 ***	48.5 ***	6.5 *	36.1 ***	36.4 ***	23 ***	23 ***	8.7 ***	17.2 ***
sympy	23.3 ***	54.2 ***	27.6 ***	19.4 ***	27.6 ***	29.5 ***	11.8 ***	14 ***	7.5 **	14 ***
node-v0.x-archive	43.8 ***	76.9 ***	21.1 ***	7 *	9.5 **	34.7 ***	65 ***	13.8 ***	3.1 *	6.5 **
wet-boew	51.3 ***	18 ***	31.1 ***	15.7 ***	28.3 ***	32.3 ***	11.7 ***	13.2 ***	7.5 ***	11.7 ***
katello	52.6 ***	107.1 ***	70.6 ***	52.6 ***	45 ***	36.4 ***	66.7 ***	45.2 ***	36.4 ***	25 ***
Average	68.2	73.9	49.3	14.3	46.4	51.4	61	30.6	10.5	29.6

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