2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

Who Will Leave the Company?

A Large-Scale Industry Study of Developer Turnover by Mining Monthly Work Report

Lingfeng Bao[*], Zhenchang Xing[†], Xin Xia[‡], David Lo[§], Shanping Li[*]
[*]College of Computer Science and Technology, Zhejiang University, China
[†]Research School of Computer Science, Australian National University, Australia
[‡]Department of Computer Science, University of British Columbia, Canada
[§]School of Information Systems, Singapore Management University, Singapore
{lingfengbao, xxia, shan}@zju.edu.cn, zhenchang.xing@anu.edu.au, davidlo@smu.edu.sg

***Abstract*—Software developer turnover has become a big challenge for information technology (IT) companies. The departure of key software developers might cause big loss to an IT company since they also depart with important business knowledge and critical technical skills. Understanding developer turnover is very important for IT companies to retain talented developers and reduce the loss due to developers' departure. Previous studies mainly perform qualitative observations or simple statistical analysis of developers' activity data to understand developer turnover. In this paper, we investigate whether we can predict the turnover of software developers in non-open source companies by automatically analyzing monthly self-reports. The monthly work reports in our study are from two IT companies. Monthly reports in these two companies are used to report a developer's activities and working hours in a month. We would like to investigate whether a developer will leave the company after he/she enters company for one year based on his/her first six monthly reports.**

To perform our prediction, we extract many factors from monthly reports, which are grouped into 6 dimensions. We apply several classifiers including naive Bayes, SVM, decision tree, kNN and random forest. We conduct an experiment on about 6-years monthly reports from two companies; this data contains 3,638 developers over time. We find that random forest classifier achieves the best performance with an F1-measure of 0.86 for *retained* developers and an F1-measure of 0.65 for *not-retained* developers. We also investigate the relationship between our proposed factors and developers' departure, and the important factors that indicate a developer's departure. We find 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 are the top three important factors.

***Keywords*-developer turnover, prediction model, mining software repositories**

I. Introduction

Software developers are the key asset of an Information Technology (IT) company. For the continuous and stable growth of the company, it is crucial to maintain a stable body of committed and experienced software developers. Unfortunately, throughout the development of an IT company, the influx and retreat of software developers, which refer to *turnover*, are very frequent. Witaker reports that up to 20% of IT software developers turnover each year [1]. Jiang and Klein find that there are almost 25% to 35% turnover rate in a study of 101 information system professionals [2]. A survey of 1,000 full-time workers conducted by the online recruitment firm Headhunter.net reports that 78% would take a new position if the right opportunity comes along and 48% of those who are employed are looking for new jobs [3].

If developer turnover is not properly handled, it may affect the success of a software project and cause significant loss for company, because software developers could depart with a lot of critical knowledge and experience. Hence, being able to predict who will leave the company early would enable the opportunity to retain the talented software developers and reduce the loss when they leave. Researchers have investigated developer turnover to understand developers' motivation of departure. Many factors can affect developer turnover, such as personal expectation [4], organizational commitment [5], developers' experience and knowledge [6], etc. Some researchers also investigate the impact of developers turnover on software quality [7], [8]. Mockus also finds a relationship between developer turnover and productivity in commercial projects [9].

However, most of existing studies are conducted in the open source communities, and only perform some qualitative observations and/or simple quantitative analysis of developers' activity data collected during software project management. Different from existing studies, in this paper we want to investigate whether we can predict the turnover of software developers in non-open-source companies by analyzing the developer activity data and leveraging data mining techniques.

There are many different kinds of developer activity data collected during software project management. In this study, the data we use is developers' monthly work reports, which are from two IT companies (named Company C1 and C2). Company C1 currently has more than 500 employees and Company C2 has more than 2000 employees. Both of them are outsourcing companies, and have a large number of projects that require different business knowledge and techniques. These two companies have established business model and rigorous project management process. They also have close collaboration with our research group so that we can access its sensitive developer data for the study. We formulate the prediction task of developer turnover as a binary classification problem based on the developers' monthly work reports.

Xin Xia is the Corresponding author

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.58

TABLE I
EXAMPLES OF MONTHLY REPORTS

	Example 1	Example 2	Example 3
Month	2011-02	2011-02	2011-10
Employee ID	1	2	3
Employee Name	D1	D2	D3
Project Name	P1	P2	P3
Tasks	1.Learn the technology about the Flex3, using Flex Builder3 and taking some exercise. 2.Reading the Screen Design of the 1580 project and the corresponding usecases. 3.Learning JSF, Primefaces and do exercise. 4.Learning about Maven, Nexus and do some exercise.	1 Disussed the use cases about the Ruby project 2 Complete UIs of the prototype project 3 Learn something about JSF.	Report Management Development
Hours	128	184	144

Monthly reports in these two companies are submitted at the end of a month, which usually contain a developer's major accomplishments or activities, and the working time in the month. A developer's monthly report must be confirmed and approved by his/her project manager. Notice that the working hours differences in monthly reports do not affect developers' salary. Project managers use monthly reports to understand the activities and workload of each member in the project, evaluate the project process in current month, and make the plan for the next month.

Table I shows three examples of monthly reports in the Company C1. Each monthly report has following fields: *month, employee id, employee name, project name, tasks and hours*. The *tasks* field refers to the description of work of a developer in a month written by the developers themselves. The content of *tasks* has no strict requirements. Hence, the writing style of *tasks* could be very different. For instance, the *tasks* field in example 1 and 2 of Table I list the developers' work in details, while the description of tasks in example 3 is very short and simple. Moreover, the content of *tasks* could be empty. The different written style of *tasks* may be due to different reasons, such as the personality of developers, their attitude towards to the work and monthly report, or the developer may fill in the monthly report with few details simply because there is too much other work to do. The *hours* field refers to the working time of a developer that he works for the project in this month. A developer could work for more than one project in one month. In this situation, the developer is required to fill a monthly report for each project. Moreover, working hour is important to a company especially an outsourcing company, since the company will use it to charge money from the clients. Also at the end of the project, project managers will count all of the working hours and check whether the costs overrun the budget.

In this paper, our goal is to predict whether a developer will leave the company or not after certain period of time based on the developers' monthly report data. Since every new developer in these two companies has a six-month probationary period, we use the first six monthly reports after the developers enter into the company to predict the developer turnover. As the two companies do not keep records of developers' departure, we consider a developer has left the company if there are no monthly reports submitted after certain time point. Therefore, our prediction task is defined as follows:
Given a developer's first six monthly report data, can we effectively predict whether the developer will leave the company or not (i.e. not-retained vs. retained) after he/she enters the company for one year?

We collect about 6 years monthly report data from the two IT companies, which contains 3638 developers and more than 400 projects. Among these 3638 developers, there are 1045 developers from Company C1 and 2593 developers from Company C2. We extract 67 features from the monthly report data which belong to six dimensions: 1) working hours of each month, 2) overall statistics of working hours, 3) statistics of task reports, 4) readability of task reports, 5) project statistics of each month, and 6) overall project statistics. Based on these extracted features, we would like to investigate the following three research questions:

RQ1: Can we effectively predict whether a developer will leave the company after he/she enters the company for one year based on monthly report data?

We apply several classic classifiers including naive Bayes, SVM, decision tree, kNN and random forest, and conduct an experiment on about 6-years monthly report data from two companies that contains 3,638 developers over time. We find that we can effectively predict whether a developer will leave the company based on monthly report data. The random forest classifier has the best performance which achieves F1-scores for predicting *retained* and *not-retained* developers of 0.86 and 0.65 on the combined dataset, respectively.

RQ2: Do the characteristics of retained and non-retained developers differ? How these relationships are different between *not-retained* and *retained* developers?

We compare the values of each factors between selected *not-retained* developers and *retained* developers by applying the Mann-Whitney U test at $p-value = 0.01$ and calculating Cliff Delta. We find that developers who leave the company are significantly different from developers who stay at the company in 31 out of the 67 factors, including the working hours of the first and the sixth month, the content of task report, and the variance of working hours.

RQ3: What are the important factors that could indicate, with high probability, that a developer will leave the company?

To compare the importance of the factors, we learn a random-forest classifier using the factors to identify whether a developer will leave or not. Correlation and redundancy analysis are applied to better model the integrated impact of factors on developers' departure. We find that the mean of the number of token in task report, the standard deviation of working hours, and the standard deviation of working hours of project members in the first month are the top three important factors in determining the likelihood of a developer's departure.

TABLE II
THE MONTHLY REPORT DATA

	Company C1	Company C2	C1 + C2
Total	1,045	2,593	3,638
Retained	699 (66.9%)	1,670 (64.4%)	2,369 (65.1%)
Not-Retained	346 (33.1%)	923 (35.6%)	1,269 (34.9%)

Paper Structure: The remainder of the paper is structured as follows. Section II describes our monthly report data and experiment setup. Section III presents the results of three research questions. Section VI reviews related work. Section VII concludes the paper and discusses future directions.

II. Case Study Setup

A. Monthly Report Dataset

We collect about six year monthly report data from our two studied companies which ranges from January 2010 to November 2015. The monthly report dataset contains more than 5,000 developers who ever submit monthly reports[1]. We exclude developers who submit less than 6 monthly reports since there is no enough data to extract features for prediction model. In our study, we extract features based on a developer's first six month reports then predict whether he/she will leave in the company in the future. We do so because a developer's probationary period is six month in these two companies. The developers who leave within 6 months do not carry much knowledge of the company and are likely to be asked to leave (due to poor performance). Finally, among them, 3,638 developers submit 6 or more monthly reports, including 1,045 developers from Company C1 and 2,593 developers from Company C2, as shown in Table II. Furthermore, these developers work for more than 400 projects, which contain different business knowledge and techniques.

To investigate whether we can predict an developer will leave the company, we divide the developers in our monthly report dataset into two groups: those who leave the company after he/she enters company in one year, and those who still stay in the company after one year. Finally, There are 1,269 developers (346 and 923 developers from Company C1 and C2, respectively) who leave the company in one year after he/she enters the company and 2,369 developers (699 and 1,670 developers from Company C1 and C2, respectively) still stay at the company after one years work.

B. Factors Potentially Affecting Developers' Departure

In this study, we consider a developer's first six month report data and extract 67 features along six dimensions, that might be correlated with developers' departure. We describe the meaning of each factor in Table III.

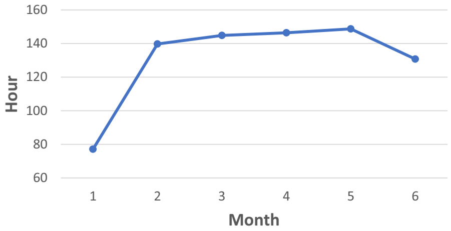

Fig. 1. The Average Reported Hours of The First Six Months

Working Hours of Each Month refers to a developer's working hours reported in the monthly report for each month. The working hours are correlated with a developer's workload. Software developers are often asked to take heavy workload and have tight deadlines. Heavy workload might be a factor which affects a developer's departure. On the other hand, if a developer's working hours are less than normal working hours, he/she might perform other non-work-related stuff and does not focus on his/her work. This might be an indicator of a developer's departure. Figure 1 shows the average reported hours of the first six month over the whole dataset. We find the average reported hours of the first month (77.15 hours) are much less than that of the other months. As discussed with the HR department of these two companies, we find since there will be many new employee training courses in the first month, thus the working hours are greatly reduced. There is also a dip from the 5th month to the 6th month, this might be because the employees also need to attend some training courses at the last part of the probationary period.

Overall Statistics of Working Hours refers to factors that are based on the overall statistics of working hours in a developer's first six month. We calculate five kinds of statistics of working hours, including the *sum*, *mean*, *median*, *standard deviation* and *maximum* of the first six month working hours for each developer.

Statistics of Task Report refers to factors that are calculated based on the text information of task report written by the developers. The written style of task report could be very different, which might indicate a developer's character and working attitude. For example, a developer, who writes the monthly report in much detail, is usually very conscientious. Otherwise, a simple task report might imply that the developer does not focus on his/her work or is dissatisfied with the work. We count the length of each monthly report (i.e. the number of characters in the report), and calculate five kinds of statistics, including the *sum*, *mean*, *median*, *standard deviation* and *maximum* of length of text of task report for each developer. Sometimes, some "lazy" developers copy the text of previous task reports or write similar task reports. So, we also tokenize and stem the text of task report, and calculate the *sum*, *mean*, *median*, *standard deviation* and *maximum* of number of distinct tokens in the monthly report for each developer. In our whole monthly report dataset, the average values of *task_len_mean* and *token_mean* are 40.12 and 6.37, respectively. This means the length of description of monthly reports is usually not very long.

Readability of Task Report refers to the ease with which a reader can understand the task report. The readability of a text is measured by the number of syllables per word and the length of sentences. Readability measures can be used to tell how many years of education a reader should have before reading the text without difficulties [10], [15]. Amazon.com uses readability measures to inform customers about the difficulty of books. We use readability features of

[1] Due to information security policy of the company, the data is sensitive, and we cannot provide the detailed number

TABLE III
FACTORS POTENTIALLY AFFECTING DEVELOPER DEPARTURE

Dimension	Factor Name	Explanation
Working Hours of Each Month	*hour{N}*	working hours of one month, N is from 1 to 6 (to indicate the 1st to 6th month)
Overall Statistics of Working Hours	*hour_sum*	the sum of the first six month working hours
	hour_mean	the average of the first six month working hours
	hour_median	the median of the first six month working hours
	hour_std	the standard deviation of the first six month working hours
	hour_max	the maximum working hours in the first six month
Statistics of Task Report	*task_len_sum*	the sum of length of text of task reports
	task_len_mean	the mean of length of text of task reports
	task_len_median	the median of length of text of task reports
	task_len_std	the standard deviation of length of text of task reports
	task_len_max	the maximum of length of text of task reports
	task_zero	the number of monthly report whose lenght of task is 0
	token_sum	the sum of the token number of task reports
	token_mean	the mean of the token number of task reports
	token_median	the median of the token number of task reports
	token_std	the standard deviation of the token number of task reports
	token_max	the maximum of the token number of task reports
Readability of Task Report	*flesch*	these metrics are used to measure the readability of a text using different formulas which are usually based on the the number of syllables per word and the length of sentences [10]–[17]
	smog	
	kincaid	
	coleman_liau	
	automated_readability_index	
	dale_chall	
	difficult_words	
	linsear_write	
	gunning_fog	
Project Statistics of Each Month	*p{N}_person*	the number of persons in the project that the developer is working for in N^{th} month , where N is from 1 to 6
	p{N}_hour_mean	the mean of working hours of project members in N^{th} month
	p{N}_hour_sum	the sum of working hours of project members in N^{th} month
	p{N}_hour_std	the standard deviation of working hours of project members in N^{th} month
	p{N}_person_change	the number of changed person compared with the previous month in N^{th} month
Overall Project Statistics	*project_num*	the number of projects which a developer works for in the first six month
	multi_project	whether the developer works for multiple projects in one month
	avg_person_change	the average changed person number in projects in the first six month
	less_zero	the number of month in which the changed person number in projects is less than 0
	equal_zero	the number of month in which the changed person number in projects is equal than 0
	larger_zero	the number of month in which the changed person number in projects is larger than 0

task report as a complementary of statistics features of task report since we think readability could also be an indicator of a developer's working attitude. In our study, we use the following nine readability measures: Flesch [10], SMOG (simple measure of gobbledygook) [11], Kincaid [12], Coleman-Liau [13], Automated Readability Index [14], Dale-Chall [15], difficult words [15], Linsear Write [16], Fog [17]. We calculate these readability measures using a python package named *textstat* [18]. In our monthly report dataset, the average value of Flesch is 98.97, which indicates very easy to read. The result is very close to other readability metrics.

Project Statistics of Each Month refers to factors that represent the information of a project which a developer is working on for each month. The working environment and other members in the project might have very important effect on a developer's working experience. For example, the good collaboration with other members in the project can improve a developer's work efficiency and experience. For each month, we calculate the following measures of the project which the developer is working for: the number of project members, the sum, mean and standard deviation of working hours of project members, and the number of changed developers. The number of project members is an indicator of project size. Small project size usually means more workload to each individual in the project. The working hours of project members could reflect the overall workload in the project. And the number of changed developers might indicate the stability of the project. The developers often prefer stay at a stable project. Notice that sometimes a developer could work for more than one project in a month. In this situation, we only consider the project on which the developer spends longest time.

Overall Project Statistics refers to factors that are based on the overall project statistics in a developer's first six months of work. We count the number of project in the first six months for each developer (project_num) and check whether a developer take part in more than one project in a month (multi_project), since the experience of working for multiple projects is different from that of working for only one project and multiple projects might mean higher workload. We also count the number of developer changed in the project which a developer works for (avg_person_change, less_zero, equal_zero, larger_zero), since the stability of the project might have impact on the working experience of a developer.

C. Prediction Model

For our monthly report data, we use our proposed factors to train a classifier to predict whether a developer will leave the company after he/she enters the company in one year. We study different classifiers which are widely used in software engineering research [19]–[23], 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) [25] 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 [26]. 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.

D. Evaluation Metric

For each developer, there would be 4 possible outcomes: a developer is classified as *not-retained* when he/she truly leaves the company in one year (true positive, TP); he/she can be classified as *not-retained* when he/she does not leave the company in one year (false positive, FP); he/she can be classified as *retained* when he/she truly leaves the company in one year (false negative, FN); or he/she can be classified as *retained* when he/she does not leave the company in one year (true negative, TN). Based on these possible outcomes, we calculate the accuracy, precision, recall, F1-score for each label to evaluate the performance of classifiers which are introduced in II-C.

Accuracy: the number of correctly classified developers (both *not-retained* and *retained*) over the total number of developers, i.e. $Acc = \frac{TP+TN}{TP+FP+TN+FN}$.

Not-Retained Precision: the proportion of developers that are correctly labeled as *not-retained* among those labeled as *not-retained* developers, i.e. $P(L) = \frac{TP}{TP+FP}$.

Not-Retained Recall: the proportion of *not-retained* developers that are correctly labeled, i.e. $R(L) = \frac{TP}{TP+FN}$.

Retained Precision: the proportion of developers that are correctly labeled as *retained* among those labeled as *retained* developers, i.e. $P(NL) = \frac{TN}{TN+FN}$

Retained Recall: the proportion of *retained* developers that are correctly labeled, i.e. $R(NL) = \frac{TN}{TP+FP}$.

F1-score: summary measure that combines both precision and recall - it evaluates if an increase in precision (recall) outweighs a reduction in recall (precision). For F-measure of *not-retained* developers, it is $F(L) = \frac{2\times P(L)\times R(L)}{P(L)+R(L)}$. And for F-measure of *retained* developers, it is $F(NL) = \frac{2\times P(NL)\times R(NL)}{P(NL)+R(NL)}$. We compare the prediction results using the F1-score, which is the harmonic mean of precision and recall. This follows the setting used in many software analytics studies [19]–[21], [27]–[31].

AUC: In addition to the F1-score, we also use the Area Under the Receiver Operating Characteristic Curve (AUC) to evaluate the effectiveness of our approach. AUC is a commonly-used measure to evaluate classification performance, and many other software engineering studies also use AUC as an evaluation metric [22], [23], [30], [32]. The larger the AUC is, the better is the performance of a classification algorithm.

III. Experiment Results

In this section, we present and discuss the answer to three research questions we proposed in Section I.

A. (RQ1) Can we effectively predict whether a developer will leave after he/she enters the company for one year based on monthly report data?

Motivation: In order to retain talented software developers and reduce the loss due to the departure of key developers, we would like to effectively predict whether developers will leave company after they enter the company for certain time period. Therefore, we use our proposed factors extracted from developers' monthly report and apply different prediction models to examine whether it is feasible to build accurate models that help to predict developers' departure.

Approach: We use the Weka tool [33] to implement these prediction models. We use 10-fold cross validation to estimate the results of these prediction models. 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 datasets is kept the same as the original dataset to simulate real-life usage of the algorithm. To evaluate their perfomance, we use accuracy, precision, recall, F1-score, and AUC metrics. The reported performance of the models is the average of 10-fold cross validation. The above approach is applied both on the monthly

TABLE IV
THE ACCURACY AND AUC OF PREDICTION MODELS

	Accuracy			**AUC**		
	Company C1	**Company C2**	**C1 + C2**	**Company C1**	**Company C2**	**C1 + C2**
Random Prediction	50.0%	50.0%	50.0%	0.50	0.50	0.50
Naive Bayes	66.6%	55.8%	57.7%	0.70	0.70	0.70
SVM	67.2%	64.7%	65.5%	0.51	0.51	0.51
Decision Tree	74.5%	71.5%	71.9%	0.74	0.68	0.68
KNN	77.5%	72.9%	74.6%	0.75	0.71	0.72
Random Forest	**81.7%**	**79.5%**	**79.7%**	**0.84**	**0.81**	**0.82**

TABLE V
PRECISION, RECALL, AND F1-SCORE ON RETAINED DEVELOPERS FOR FIVE PREDICTION MODELS

	Company C1			**Company C2**			**C1 + C2**		
	Precison	**Recall**	**F1**	**Precison**	**Recall**	**F1**	**Precison**	**Recall**	**F1**
Random Prediction	0.67	0.50	0.57	0.64	0.50	0.56	0.65	0.50	0.57
Naive Bayes	0.81	0.65	0.72	0.82	0.40	0.54	**0.82**	0.45	0.58
SVM	0.67	**0.99**	0.80	0.65	**0.99**	0.79	0.65	**0.99**	0.79
Decision Tree	0.81	0.81	0.81	0.77	0.79	0.78	0.78	0.80	0.79
KNN	**0.83**	0.83	**0.83**	0.79	0.79	0.79	0.80	0.81	0.81
Random Forest	0.81	0.94	0.79	**0.93**	0.85	**0.87**	0.79	0.93	**0.86**

TABLE VI
PRECISION, RECALL, AND F1-SCORE ON NOT-RETAINED DEVELOPERS FOR FIVE PREDICTION MODELS

	Company C1			**Company C2**			**C1 + C2**		
	Precison	**Recall**	**F1**	**Precison**	**Recall**	**F1**	**Precison**	**Recall**	**F1**
Random Prediction	0.33	0.50	0.40	0.36	0.50	0.42	0.35	0.50	0.41
Naive Bayes	0.50	**0.70**	0.58	0.44	**0.84**	0.58	0.44	**0.81**	0.57
SVM	0.80	0.01	0.02	0.79	0.01	0.02	**0.88**	0.01	0.02
Decision Tree	0.62	0.60	0.61	0.61	0.58	0.59	0.60	0.58	0.59
KNN	0.66	0.66	0.66	0.62	0.62	0.62	0.64	0.63	0.63
Random Forest	**0.83**	0.57	**0.67**	**0.81**	0.55	**0.66**	0.81	0.55	**0.65**

report data from Company C1 and Company C2. Furthermore, we combine the two companies' monthly report data into a single dataset, then apply the same approach on it. We also choose a baseline model, i.e. random prediction, to compare our proposed prediction models. In random prediction, it randomly predicts developers' departure. The precision for random prediction is the percentage of *not-retained* or *retained* developers in the data set. Since the random prediction model is a random classifier with two possible outcomes (e.g., *not-retained* or *retained* developers), its accuracy, AUC, and recall are 0.50.

Results: Table IV shows the accuracy and AUC of the five different prediction models we apply. We find that the accuracies and AUCs of all 5 classifiers are larger than the baseline model, i.e. random prediction. For the results of accuracy, we find that the accuracies of 3 out of 5 classifiers, i.e., decision tree, KNN and random forest, are larger than 70% on all of three datasets. Moreover, the random forest classifier has the highest accuracy on all of three datasets. For the results of AUC, we also find the random forest achieves the highest AUC among these five classifiers. The AUCs of the random forest on these three datasets are all larger than 0.8, while all AUCs of other four classifiers are less than 0.8. We also find that the results of accuracy and AUC on the two company datasets and the combined dataset is very close. From Table IV, we can see the random forest achieves the best overall performance.

Table V shows the precision, recall, F1-score of the baseline model and these 5 classifiers for predicting *retained* developers. The number with bold text in the table is the largest for each column. We find all classifiers have good performance on the dataset of Company C1, i.e. all F1-scores are larger than 0.7, and are also much larger than that of the baseline model. On the dataset of Company C2 and the combined dataset, all classifiers also have very good performance except Naive Bayes, since the recall of Naive Bayes is very low, only 0.45. All other classifiers' F1-scores for predicting *retained* developers are very high, which are close to or higher than 0.80, which is much larger than that of the baseline model. In short, we think the random forest classifier achieves the best performance, since it has the highest F1-score on the dataset of Company C2 and the combined dataset, i.e. 0.87 and 0.86, respectively, and its F1-score on the dataset of Company C1 is 0.79, which is very close to the highest F1-score, 0.83, which is from KNN classifier.

Table VI shows the precision, recall, F1-score of the baseline model and these 5 classifiers for predicting *not-retained* developers. We find that the results of these 5 classifiers for predicting *not-retained* developers on the two company dataset and the combined dataset is very similar. The random forest classifier also has the best performance which has the highest F1-scores on all the three dataset. While the SVM classifier has very bad performance as its recalls are very low, which are all lower than 0.1. The F1-scores of other 3 classifiers are close to or higher than 0.60, while the F1-score of the baseline model is only about 0.40. In summary, we find that the random forest has the best performance on predicting whether a developer will leave the company after he/she enters the company in one year or not.

To measure whether the improvement of these 5 classifiers over random prediction is statistically significant, we apply

TABLE VII
RELATIONSHIP BETWEEN FACTORS AND DEVELOPERS' DEPARTURE WITH SIGNIFICANT DIFFERENCE

Dimension	Factor Name	Rel.	d-value
Working Hours of Each Month	hour1	-	0.194
	hour6	-	0.153
Overall Statistics of Working Hours	hour_sum	-	0.192
	hour_mean	-	0.192
	hour_std	+	0.290
Statistics of task report	task_len_sum	-	0.317
	task_len_mean	-	0.320
	task_len_median	-	0.294
	task_std	-	0.255
	task_len_max	-	0.297
	task_zero	+	0.312
	token_sum	-	0.305
	token_mean	-	0.308
	token_median	-	0.280
	token_std	-	0.245
	token_max	-	0.291
Readability of task report	flesch	+	0.283
	kincaid	-	0.287
	coleman_liau	-	0.313
	automated_readability_index	-	0.292
	dale_chall	-	0.313
	difficult_words	-	0.297
	linsear_write	-	0.298
	gunning_fog	-	0.289
Project Statistics of Each Month	p1_hour_mean	-	0.148
	p1_hour_std	+	0.247
	p2_hour_std	+	0.192
	p3_hour_std	+	0.188
	p4_hour_std	+	0.187
	p5_hour_std	+	0.164
	p6_hour_std	+	0.190

Wilcoxon signed-rank test [34] at 95% significance level on 10 folds of F1-scores. We also use Bonferroni correction [35] to counteract the results of multiple comparisons. We find all p-values are smaller than 0.05 which indicates the improvement is statistically significant at the confidence level of 95%.

We can effectively predict whether a developer will leave the company based on monthly report data. The random forest classifier has the best performance which achieves F1-score for retained and not-retained developers of 0.86 and 0.65 on the combined monthly report data, respectively.

B. (RQ2) Do the characteristics of retained and non-retained developers differ? How these relationships are different between retained and not-retained developers?

Motivation: We have proposed 67 factors that could potentially affect a developer's departure. In this research question, we are interested in investigating how each factor is related with the developers' departure. The company leader and manager can use the results of this question to understand whether the characteristics of retained and non-retained developers differ and take proactive actions.

Approach: We compare the values of each factors between selected *not-retained* developers and *retained* developers. We first analyze the statistical significance of the difference between the two groups of developers, i.e. *not-retained* developers and *retained* developers, by applying the Mann-Whitney U test at $p - value = 0.01$. To show the effect size of the difference between the two groups, we calculate Cliff Delta, which is a non-parametric effect size measure. Cliff defines a delta of less than 0.147, between 0.147 to 0.33, between 0.33 and 0.474, and above 0.474 as negligible, small, medium, and large effect size, respectively.

Results: Table VII shows the factors that have $p - value < 0.01$ and $d > 0.147$ (i.e., statistically significant difference with at least a small effect size). In Table VII, the column Rel. is short for relationship, "+" means developers who leave the company have significantly higher value on this factor while "-" means developers who stay in the company have significantly higher value on this factor. We find that the *not-retained* and *retained* developers have significant difference in 31 out of the 67 factors. The effect size is small for all the factors. Note that relationship between factors and developers' departure with non-significant difference is not included in the table.

For working hours of each month dimension, the working hours of the first and sixth month can differentiate *not-retained* developers from *retained* developers. This suggests that the company might pay attention to the work of developers in the first month and the last month of probation period. For overall statistics of working hours dimension, the *retained* developers have significantly higher value on factors **hour_sum** and **hour_mean** while the *not-retained* developers have significantly higher value on factor **hour_std**. We find that the total working hours of *not-retained* developers are less than those of *retained* developers and the variance of working hours of *not-retained* developers is larger than that of *retained* developers.

For statistics of task report dimension and readability of task report dimension, most of the factors have statistically significant difference except **smog**. Notice that the lower value of **flesch** means more difficult to read, which is opposite from all other readability metrics. So, only **flesch** shows positive correlation with developers' departure. *Not-retained* developers usually write less content of task and are more prone to submit empty monthly report than *retained* developers. The readability of task report of *not-retained* developers is often worse than those of *retained* developers.

For project statistics of each month dimension, the average working hours of project members in the first month can differentiate *not-retained* developers from *retained* developers. Furthermore, *not-retained* developers have significantly higher value on the variance of working hours of project members for each month. This might suggest us that the working environment has big impact on developers' departure. If the workload among project members is very different, the turnover of the project may increase. Therefore, project managers should pay attention to balance the workload between project members.

Not-retained developers are significantly different from retained developers in 31 out of 67 factors. Generally, not-retained developers have less working hours in the first and the sixth month (i.e., the start and end of the probation period), write less content in task report of monthly reports. Moreover, there is larger variance of working hours of project members in their projects for not-retained developers.

TABLE VIII
GROUPS OF VARIABLES THAT HAVE CORRELATIONS LARGER THAN 0.7

Group
hour_median, hour_sum, **hour_mean**
linsear_write, token_sum, token_std, task_len_mean, **token_mean**, token_median, automated_readability_index, dale_chall, task_len_std, coleman_liau, task_len_sum, task_len_max, kincaid, task_len_median, flesch, difficult_words, token_max, gunning_fog, task_zero
p5_hour_sum, p6_person, p6_hour_sum, p2_person, p3_person, p4_person, p5_person, p2_hour_sum, p4_hour_sum, p3_hour_sum

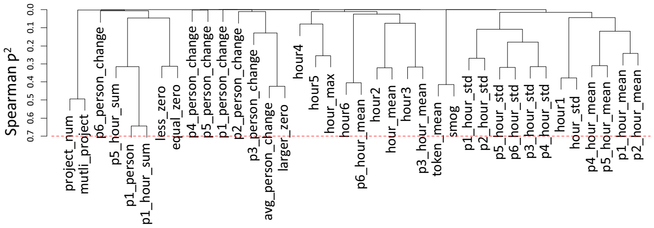

Fig. 2. Cluster factors after correlation analysis

C. (RQ3) What are the important factors that could indicate, with high probability, that a developer will leave the company?

Motivation: Although a developer's departure is impacted by multiple factors, some factors might be more influential on developers' departure than others, which need more attention. Therefore, in this research question, we would like to find out these important factors. Companies can use these important factors to make proactive plan to retain developers and prevent risks which are caused by developers' departure.

Approach: Random forest has been proved to have the best performance on predicting a developer's departure in RQ1. Therefore, we only consider to identify the most important factors in the random forest model. Comparing with the random forest model in RQ1, we first perform variable selection to build another random forest classifier because correlated variables might lead to poor models which are hard to interpret [36]. Variable selection process contains two steps:

Step 1: Correlation Analysis. We use a variable clustering analysis, implemented in a R package named **Hmisc**, to construct a hierachical overview of the correlations among the factors. For sub-hierarchies of factors with correlation larger than 0.7, which is the same setting to a previous study [37], we select only one variable from the sub-hierarchy for inclusion in our model. In particular, out of 67 factors, there are 32 factors which are belong to three groups of variables that have correlations larger than 0.7 (see Table VIII). We use three variables and remove other variables from the relevant group, i.e. **hour_mean**, **token_mean**, **p5_hour_sum**. There are 38 factors left after correlation analysis. The final hierarchical overview is presented in Figure 2.

Step 2: Redundancy Analysis. Correlation analysis reduces collinearity among the factors, but it may not detect all of redundant factors, i.e., factors that do not have a unique signal relative to the other factors. Redundant factors in an explanatory model will interfere with one another, distorting the modeled relationship between the factors and predictors. We remove redundant factors by using the implementation provided by the **redun** function in the **rms** R package. In particular, from the leftover 38 factors through correlation analysis, we remove **hour4**, **equal_zero** because they can be represented using other factors.

After two steps of removing redundant variables, we have 36 remaining factors. Then we use the **bigrf** R package to implement the random forest classifier. We also use 10-fold cross validation to evaluate the effectiveness of the model. To identify the important factors in determining developers' departure, we use the **varimp** function in **bigrf** package to compute the importance of a factor in training process based on out of the bag (OOB) estimates, which is an internal error estimate of a random forest classifier [38]. The underlying idea is to permute each factor randomly one by one and see whether the OOB estimates will be reduced significantly or not.

For each run of the 10-fold cross validation, we get an importance value for each factor. To determine the factors that are the most important for the whole dataset, we take the important values from all 10 runs and apply the Scott-Knott test [39]. 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.

Results: The 10-fold cross validation shows the random forest using selected factors can infer *retained* developers with an F1-measure of 0.857 and *not-retained* developers with an F1-measure of 0.664. This result is similar with that of the random forest using all proposed factors in RQ1.

Figure 3 shows the Scott-Knott test results when comparing the importance value of factors. Different groups of variable whose importance values are statistically significant different from other groups of variables ($p{-}value < 0.05$). We find that the mean of the number of token in task report (token_mean), the standard deviation of working hours (hour_std), and the standard deviation of working hours of project members in the first month (p1_hour_std) are the top three important factors that influence the random forest model. This result is consistent with the results in RQ2 (see section III-B). The effect size of these three factors is bigger than that of other factors (see Table VII).

Notice that we use the factor token_mean to represent factors that are related to task report but removed in correlation analysis step. The more content written in the task report of monthly report, the more likely a developer stay in the company. The detailed task report might indicate that a developer is more conscientious on work and prefers to stay in the company and grows up with the company together. Larger variance of working hours may imply the working state of the developer is not very stable and such kinds of developers are more likely to leave the company. Moreover, the imbalance of working hours of project members in the first month might leave a bad first impression on the project and the company. This implies that working environment has big impact on developer turnover. These factors help most in determining the developer who leaves the company after he/she enters the company for one year.

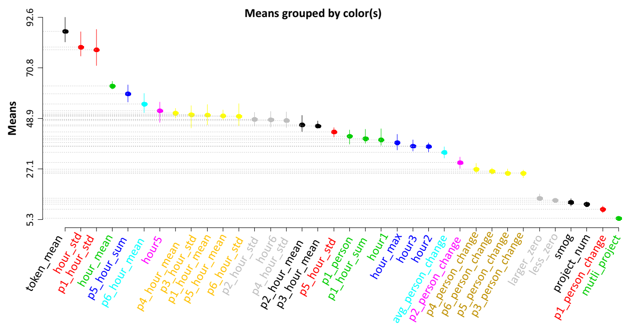

Fig. 3. Scott-Knott test results when comparing the importance values of factors, divided into distinct groups that have a statistically significant difference in the mean ($p - value < 0.05$)

The mean of the number of token in task report, the standard deviation of working hours, and the standard deviation of working hours of project members in the first month are the top three important factors in determining the likelihood of a developer's departure

IV. Threats to Validity

Threats to internal validity refer to errors in our code and experiment bias. We use the default setting in Weka to train our classifiers, which is similar to the setting of prior work [30]. We also double check our code, however, there may exist some errors that we do not notice. Moreover, to mitigate the bias of results selection, we run the 10-fold cross-validation and present the average performance. Another threat to internal validity is the setting that we use the first six monthly reports to predict a developer's departure after he/she enters the company for one year. This is because every developer in the studied companies has a six-month probationary period and most of them at least submit six monthly reports. Admittedly, excluding developers who do not submit 6 monthly reports might skew the dataset.

Threats to external validity relate to the generalizability of our findings. In this study, our monthly reports are from two IT companies. Thus, it is unclear whether the same results still hold for other developers from other companies. However, we analyze more than 3,000 developers' monthly reports and these developers belong to more than 400 projects which use different programming languages and business knowledge. Moreover, monthly report is a common practice and often used in project management.

Another threats to external validity relate to the generalizability of our extracted features. In this paper, we extract 67 factors which are categorized into 6 dimensions from two independent companies. These two companies use different monthly reporting systems, and we focused on the general factors, and tried to avoid to extract the specific factors which only exist to a specific monthly reporting system. After we extracted the factors, we also discuss with the people in the HR departments[2] whether these factors are reasonable and available to other IT companies. And all of them agree that it is possible to extract the similar features from other IT companies. Regarding the factors that we have considered, there might be additional factors that could be more relevant to developers' departure. In the future, to reduce these threats, we plan to investigate more developers from different companies and consider more factors.

Threats to construct validity refer to the suitability of our evaluation measures. We use F1 and AUC scores which are also used by prior studies to evaluate the effectiveness of various software engineering studies [19]–[23]. Thus, we believe there is little threat to construct validity.

V. Discussion

First, previous studies often use survey or interview to investigate the factors that have impact on developer turnover [40]. In this study, we show that data-driven approach can be used to effectively predict developer turnover by mining some kinds of activity data of developers, i.e. monthly report of developers. Comparing with previous studies that use survey or interview, the data-driven approach is more quantitative and objective.

Second, our findings might be complementary of previous studies. The factors found in our study is low level and easily calculated. The high level factors considered in prior studies (e.g. person expectation, job satisfaction, etc) are usually hard to measure. Moreover, we believe our low level factors may reflect some high level factors; for example, more working hours, which usually means more pressure on developers, is like to decrease job satisfaction of developers. In the future, we will investigate the connection between our low level factors and high level factors. Furthermore, we believe that more data from developers' daily work can be used to extract more features to help us study developer turnover and understand the underlying motivations of developers staying or leaving a company. Such daily work data can be tracked using software application instrumentation methods [41]–[45]. For example, MYLAR (currently referred to as MYLYN) listens to Eclipse IDE selection and view services to monitor programmer activities in the Eclipse workbench [46]. ActivitySpace can unobtrusively tracks developer interaction with the working environment during the work [47]. We can use such kinds of developer interaction data for various studies on developer behavior and company management, including the study of developer turnover. We can also investigate the behavior of developers who are predicted to leave the company using more interaction data.

Third, monthly report is a very common practice in many IT companies, however, there are limited investigations on mining monthly report. Our paper presents the first study on how to leverage knowledge behinds monthly reports to predict developer turnover. We hope our study can inspire more researches on mining monthly reports. Also, based on monthly reports, we can also develop other automated tools, e.g., we can use topic modelling [48] to analyze the development trends of a project, and we can even predict whether a project will be successful by aggregating project members' factors extracted from monthly reports.

[2] Some HRs worked in multiple IT companies before they join C1 or C2.

Finally, our tool can also help to reduce the potential risks in a project team, and help to detect and train potential excellent employees. For example, if our tool predict that a developer will leave the company, his/her project manager can: (1) encourage and communicate with him/her frequently, and increase the salary or bonus to retain the developer (if he/she has much potential and is hardworking); (2) involve another developer as a backup, to avoid the risk due to the developer resignation. On the other hand, if a developer is predicted to stay at the company, the PM could consider to give him/her more resources (e.g., increase salary or bonus) and train him/her to be a potential leader of a project team.

Besides, project manager can use the results of our tool as a complement to productivity measures, since developers who want to leave the company may not focus on their work and tend to write poor quality code. Project manager can give more training to developers who are predicted to stay in a team, and care less about developers who are predicted to leave the company.

VI. Related Work

In this section, we briefly review the related works on the reasons and impact of developer turnover.

A. Reasons of Developer Turnover

Researchers have developed a number of significant theoretical models to better understand employee turnover, such as Price-Mueller model [49] and Jackofsky and Slocum's integrated process model [50]. According to Mobley [51], the determinants of employee turnover can be simplified into four general classes: 1) the external economy, which affects the availability of alternative jobs; 2) organizational factors, such as leadership, the reward system, and job design; 3) individual non-work variables, like a spouse's career and family considerations; 4) individual work-related variables, such as values, expectations, abilities, satisfaction, commitment, and intentions.

A lot of studies have been conducted to understand developer turnover in software engineering community. Yu *et al.* find that the objective attribute of open source software (OSS) project and personal expectations are the two most important factors to predict turnover [4]. Schilling *et al.* analyze the contribution behavior of former Google Summer of Code and report that the level of development experience and conversational knowledge is strongly associated with developer retention [6]. Hynninen *et al.* conduct a survey with developers and find that developer turnover can be an important manifestation of low commitment [5]. Sharma *et al.* consider both the developer and project level factors and suggest that past activity, developer role, project size and project age are important predictors of turnover [52]. Different from previous studies which mainly focus on open source projects, we extract a number of factors from monthly work reports which are used for project management in an industrial company and use data mining technique to predict developer turnover.

B. Impact of Developer Turnover

Obviously, employee turnover could cause economic loss to companies. Pekala reports that firms in the U.S pay more than $140 billion annually in recruiting, training, and administrative expenses to replace employees who leave [3]. In online communities and collaborative platforms, such as Wiki projects, the departure of contributors has a negative effect on the community and causes social capital losses [53]. However, employee turnover could be a good opportunity for organizations, as leavers are those most dissatisfied with the current organization, and those who remain enjoy better conditions and performance [54]. Moderate levels of turnover could bring fresh level of activities, novel knowledge, and liveliness for the online communities [55], [56].

Developer turnover could cause knowledge loss in software development group. Izquierdo-Cortazar *et al.* propose some measures of knowledge loss [57], such as the evolution of orphan lines of code lastly edited by a developer who left the team. Fronza *et al.* propose a wordle to visualize the level of cooperation of a team and mitigate the knowledge loss due to turnover [58]. According to a survey conducted by Hall *et al.* [59], developer turnover might be related to project success. Developer turnover also has impact on software quality [7], [8]. Mockus finds that only leavers have relationship with software quality since the loss of knowledge and experience [7]. On the contrary, Foucault *et al.* find that newcomers have a relationship with quality and leavers do not have such relationship [8].

VII. Conclusion

In this paper, based on large-scale monthly work reports from two IT companies, we use data mining technique to investigate whether a developer will leave the company after he/she enters into the company for one year. The monthly reports we used are submitted by 3,638 developers in about 6 year period. Our study reveals the most effective classifier (i.e., random forest) for the prediction of developers' departure. Our study also identifies the key relationship between various dimensions of factors and developers' departure, and the important factors which indicate a developer's departure. The model from our work can potentially help a company to effectively predict the potential of a developer's departure and take proactive actions to retain talented developers; for example, by better managing workload variance among project members. In the future, we will collect more monthly reports to verify our approach. We will also consider more activity data from developers and extract more factors to investigate developer turnover in more IT companies.

Acknowledgment

This research was supported by NSFC Program (No. 61602403 and 61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2015BAH17F01).

References

[1] A. Whitaker, “What causes it workers to leave,” *Management Review*, vol. 88, no. 9, p. 8, 1999.

[2] J. J. Jiang and G. Klein, “Supervisor support and career anchor impact on the career satisfaction of the entry-level information systems professional,” *Journal of management information systems*, vol. 16, no. 3, pp. 219–240, 1999.

[3] N. Pekala, “Holding on to top talent,” *Journal of Property management*, vol. 66, no. 5, pp. 22–22, 2001.

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

[5] P. Hynninen, A. Piri, and T. Niinimaki, “Off-site commitment and voluntary turnover in gsd projects,” in *IEEE International Conference on Global Software Engineering*. IEEE, 2010, pp. 145–154.

[6] 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 *Proc. HICSS*. IEEE, 2012, pp. 3446–3455.

[7] A. Mockus, “Succession: Measuring transfer of code and developer productivity,” in *Proc. ICSE*. IEEE, 2009, pp. 67–77.

[8] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, “Impact of developer turnover on quality in open-source software,” in *Proc. FSE*. ACM, 2015, pp. 829–841.

[9] A. Mockus, “Organizational volatility and its effects on software defects,” in *Proc. FSE*. ACM, 2010, pp. 117–126.

[10] R. F. Flesch, *How to write plain English: A book for lawyers and consumers*. Harpercollins, 1979.

[11] G. H. Mc Laughlin, “Smog grading-a new readability formula,” *Journal of reading*, vol. 12, no. 8, pp. 639–646, 1969.

[12] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, “Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel,” DTIC Document, Tech. Rep., 1975.

[13] M. Coleman and T. L. Liau, “A computer readability formula designed for machine scoring.” *Journal of Applied Psychology*, vol. 60, no. 2, p. 283, 1975.

[14] R. Senter and E. A. Smith, “Automated readability index,” DTIC Document, Tech. Rep., 1967.

[15] E. Dale and J. S. Chall, “A formula for predicting readability: Instructions,” *Educational research bulletin*, pp. 37–54, 1948.

[16] “Linsear write,” http://www.csun.edu/~vcecn006/read1.html#Linsear.

[17] R. Gunning, “{The Technique of Clear Writing},” 1952.

[18] “textstat,” https://pypi.python.org/pypi/textstat.

[19] T. Jiang, L. Tan, and S. Kim, “Personalized defect prediction,” in *Proc. ASE*. IEEE, 2013, pp. 279–289.

[20] S. Kim, E. J. Whitehead Jr, and Y. Zhang, “Classifying software changes: Clean or buggy?” *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, 2008.

[21] J. Nam, S. J. Pan, and S. Kim, “Transfer defect learning,” in *Proc. ICSE*. IEEE Press, 2013, pp. 382–391.

[22] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, “Predicting the severity of a reported bug,” in *Proc. MSR*. IEEE, 2010, pp. 1–10.

[23] 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*, vol. 34, no. 4, pp. 485–496, 2008.

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

[25] 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*, vol. 14, no. 1, pp. 1–37, 2008.

[26] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[27] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou, “Automatic, high accuracy prediction of reopened bugs,” *Automated Software Engineering*, vol. 22, no. 1, pp. 75–109, 2015.

[28] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, “Hydra: Massively compositional model for cross-project defect prediction,” *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[29] X. Xia, D. Lo, X. Wang, and X. Yang, “Collective personalized change classification with multiobjective search,” *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1810–1829, 2016.

[30] X. Xia, E. Shihab, Y. Kamei, D. Lo, and X. Wang, “Predicting crashing releases of mobile applications,” in *Proc. ESEM*. ACM, 2016, p. 29.

[31] X. Xia, D. Lo, E. Shihab, and X. Wang, “Automated bug report field reassignment and refinement prediction,” *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1094–1113, 2016.

[32] X. Yang, D. Lo, X. Xia, and J. Sun, “Condensing class diagrams with minimal manual labeling cost,” in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 22–31.

[33] 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*, vol. 11, no. 1, pp. 10–18, 2009.

[34] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[35] H. Abdi, “Bonferroni and šidák corrections for multiple comparisons,” *Encyclopedia of measurement and statistics*, vol. 3, pp. 103–107, 2007.

[36] M. N. Audris Mockus and H. Sharp, “Best practices and pitfalls for statistical analysis of se data,” in *Proc. ICSE*. IEEE, 2014.

[37] 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 *Proc. ICSME*. IEEE, 2015, pp. 301–310.

[38] D. H. Wolpert and W. G. Macready, “An efficient method to estimate bagging’s generalization error,” *Machine Learning*, vol. 35, no. 1, pp. 41–55, 1999.

[39] “Scott-knott test,” https://cran.r-project.org/web/packages/ScottKnott/ScottKnott.pdf.

[40] S. G. Westlund and J. C. Hannon, “Retaining talent: Assessing job satisfaction facets most significantly related to software developer turnover intentions,” *Journal of Information Technology Management*, vol. 19, no. 4, pp. 1–15, 2008.

[41] T.-H. Chang, T. Yeh, and R. Miller, “Associating the visual representation of user interfaces with their internal structures and metadata,” in *Proc. UIST*, 2011, pp. 245–256.

[42] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker, “TaskTracer: a desktop environment to support multi-tasking knowledge workers,” in *Proc. IUI*, 2005, p. 75.

[43] E. Harpstead, B. A. Myers, and V. Aleven, “In search of learning: facilitating data analysis in educational games,” in *Proc. CHI*, 2013, p. 79.

[44] W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless, “Edit wear and read wear,” in *Proc. CHI*, 1992, pp. 3–9.

[45] J. H. Kim, D. V. Gunn, E. Schuh, B. C. Phillips, R. J. Pagulayan, and D. Wixon, “Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems,” in *Proc. CHI*, 2008, pp. 443–451.

[46] M. Kersten and G. C. Murphy, “Mylar: a degree-of-interest model for IDEs,” in *Proc. AOSD*, 2005, pp. 159–168.

[47] L. Bao, D. Ye, Z. Xing, X. Xia, and X. Wang, “Activityspace: a remembrance framework to support interapplication information needs,” in *Proc. ASE*. IEEE, 2015, pp. 864–869.

[48] D. M. Blei, “Probabilistic topic models,” *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[49] J. L. Price, “Reflections on the determinants of voluntary turnover,” *International Journal of manpower*, vol. 22, no. 7, pp. 600–624, 2001.

[50] E. F. Jackofsky and J. W. Slocum, “A causal analysis of the impact of job performance on the voluntary turnover process,” *Journal of Organizational Behavior*, vol. 8, no. 3, pp. 263–270, 1987.

[51] W. H. Mobley, “Employee turnover: Causes, consequences, and control,” 1992.

[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*. Springer, 2012, pp. 331–337.

[53] X. Qin, M. Salter-Townshend, and P. Cunningham, “Exploring the relationship between membership turnover and productivity in online communities,” *arXiv preprint arXiv:1401.7890*, 2014.

[54] D. Krackhardt and L. W. Porter, “When friends leave: A structural analysis of the relationship between turnover and stayers’ attitudes,” *Administrative science quarterly*, pp. 242–261, 1985.

[55] S. Ransbotham and G. C. Kane, "Membership turnover and collaboration success in online communities: Explaining rises and falls from grace in wikipedia," *MIS Quarterly-Management Information Systems*, vol. 35, no. 3, p. 613, 2011.

[56] L. Dabbish, R. Farzan, R. Kraut, and T. Postmes, "Fresh faces in the crowd: turnover, identity, and commitment in online groups," in *Proc. CSCW*. ACM, 2012, pp. 245–248.

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

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

[59] T. Hall, S. Beecham, J. Verner, and D. Wilson, "The impact of staff turnover on software projects: the importance of understanding what makes software practitioners tick," in *Proceedings of the 2008 ACM SIGMIS CPR conference on Computer personnel doctoral consortium and research*. ACM, 2008, pp. 30–39.