Should I contribute to this discussion?

Walid M. Ibrahim, Nicolas Bettenburg, Emad Shihab, Bram Adams, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
{walid, nicbet, emads, bram, ahmed}@cs.queensu.ca

***Abstract*—Development mailing lists play a central role in facilitating communication in open source projects. Since these lists frequently host design and project discussions, knowledgeable contribution to these discussion threads is essential to avoid miscommunication that might slow-down the progress of a project. However, given the sheer volume of emails on these lists, it is easy to miss important discussions. To find out how developers are able to deal with mailing list discussions, we study the main factors that encourage developers to contribute to the development mailing lists. We develop personalized models to automatically identify discussion threads that a developer would contribute to based on his previous contribution behavior. Case studies on development mailing lists of three open source projects (Apache, PostgreSQL and Python) show that the average accuracy of our models is 89–85% and that the models vary significantly between different developers.**

I. Introduction

The popularity of Open Source Software (OSS) projects keeps on growing rapidly. This popularity has been driven by the voluntary efforts of thousands of developers in different locations and time zones across the globe. Many projects use mailing lists and Internet Relay Chat (IRC) channels to effectively communicate and plan their development work [1]. Messages on development mailing lists discuss important issues such as current development plans, maintenance requests, user support, bugs, design decisions and development schedules. Such information forms the basis for knowledge transfer to new developers in a project. In general, the development mailing list is where the development team lives and communicates [2].

Timely contributions help steer important development discussions in the right direction. Such timely contributions provide needed information and shed light into undocumented design issues or misunderstood requirements. However, developers of large open source projects are constantly flooded by emails on a daily basis (for example, 1,150 messages per month for the PostgreSQL project). Over-strained project members must wade through this flood of emails to decide which emails require their participation and contribution. To cope with all these emails, developers might skim emails quickly or even completely ignore them if the subject line does not catch their attention. Thus, developers are likely to miss contributing to relevant discussion threads.

In this paper, we are interested in understanding the factors that influence developers to contributing to mailing list discussion threads. For example, contributors in mailing list might reply to short messages and threads, or threads posted by the people they know. Developers typically also have particular moments during the day when they check their emails, and periods during the week when they are not available. Of course, the actual content of a message provides a definitive answer to the question whether one should contribute to a thread. Using these factors, we want to identify what motivates developers to contribute to threads in highly active mailing lists.

Based on the developers' history of thread contributions, we develop for each developer a personalized model that can identify, based on her previous contribution to a mailing list which threads require developer contribution. This model helps to understand the factors that influence the contribution of a developer. Through a case study based on the mailing lists of three open source projects (Apache, PostgreSQL and Python), we use our model to answer two research questions:

Q1 **Can we build a high accuracy model of developer contribution to a thread?**
For each developer, we build a composite model based on Naive Bayesian and Decision Tree classifiers that determines with high accuracy the contribution of developers to a thread based on her previous contribution behavior.

Q2 **What are the most important factors that influence a developer to contribute to a thread?**
Based on our composite model, we find that message content, the current length of a discussion thread and recent activity of a developer are the most important factors. We also find that the important factors vary between developers.

Our composite model combines the strengths of Naive Bayesian and Decision tree classifiers to improve the overall performance. We use a Naive Bayesian classifier to deal with the body and subject of a thread, as selecting important threads for a developer to contribute to is similar to detecting emails that are not spam. We use a Decision Tree classifier to explain the contribution behavior of a developer based on the output of the Naive Bayesian classifier along with other factors.

Organization of the Paper. Section II explains the methodology we used to build our composite model. Section III presents our data collection and thread reconstruction process. Section IV answers the two research questions using a case study. Section V discusses our findings. Section VI presents threats to validity. Section VII discusses related work. Finally, Section VIII concludes the paper.

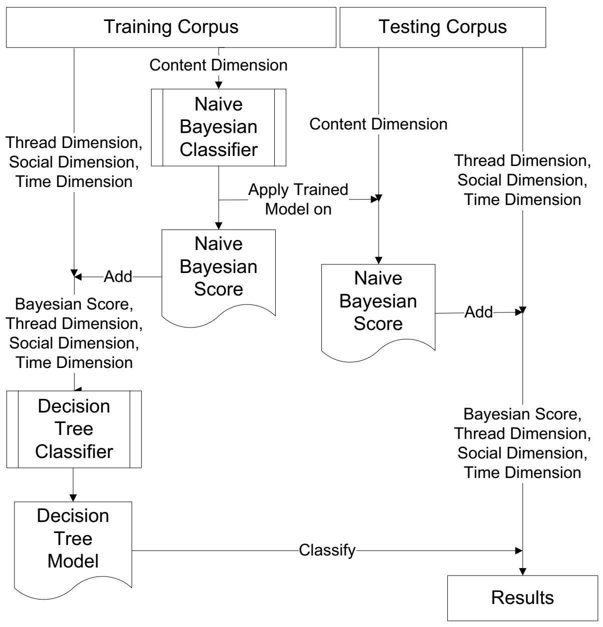
Training Corpus
Testing Corpus
Content Dimension
Naive Bayesian Classifier
Thread Dimension, Social Dimension, Time Dimension
Content Dimension
Thread Dimension, Social Dimension, Time Dimension
Apply Trained Model on
Naive Bayesian Score
Add
Naive Bayesian Score
Add
Bayesian Score, Thread Dimension, Social Dimension, Time Dimension
Decision Tree Classifier
Bayesian Score, Thread Dimension, Social Dimension, Time Dimension
Decision Tree Model
Classify
Results

Fig. 1. Our composite model building approach.

II. Methodology

Our goal is to determine the main factors that drive developers to contribute to a particular discussion thread in practice. To identify these main factors, we build a composite model that determines which discussion threads a developer would contribute to based on her previous contribution. If the model returns “do not contribute” for a developer and a given thread, this means that the developer decided not to contribute to the thread (although he might have read the thread). The decision to contribute or not might depend on various factors. The developer might not have the knowledge needed to help him to move the discussion forward, or it could be that another developer has already contributed the right information to the discussion. Our model could be considered as a type of spam filter, except that our model flags interesting threads that need developer contribution, instead of flagging threads that a developer should not read as spam messages.

We use a data mining approach to build a composite model that explains, given a corpus of historical thread discussions: Can we determine whether or not a developer should contribute to a given thread? In this section, we present the data mining elements that we need to build such a model:

1) The possible factors that can influence the decision of a developer to contribute to a given thread.
2) The composite data mining approach that we use.
3) The metrics to evaluate the accuracy of our model.

A. Contribution Factors

A developer decides to contribute to a given discussion based on various factors. Table I shows the different factors used in our approach. These factors span the following four different dimensions:

1) **The Thread dimension** consists of factors that measure the characteristics of a discussion thread. The factors that we consider are the length of the discussion (thread_length) and the size of the thread (word_count). This dimension measures the **necessity** for a developer to contribute to a discussion thread.
2) **The Social dimension** consists of factors that capture the communication activity between developers. The factors that we use are whether the person who started the discussion is well known (one of the top 20 developers) by others (known_starter), whether the person who last posted to the discussion is well known (one of the top 20 developers) by others (known_poster), or whether the developer has been active recently on the mailing list (contribution_activity). This dimension measures the impact of **inter-personal** relations such as friendship on mailing list contribution.
3) **The Time dimension** consists of factors that indicate when the message in a thread was posted. The factors we use are the message time (msg_time), day (msg_day), and month (msg_month). This dimension measures the **availability** of the developer to contribute to threads.
4) **The Content dimension** consists of factors that are related to the content of a thread. Examples of such factors are the words in the thread subject or content. This dimension captures the **expertise** of a developer on the topics that are discussed.

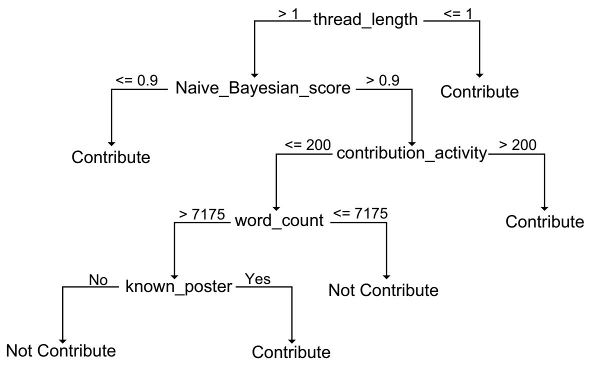

Fig. 2. A Decision Tree for Thomas Lockhart.

B. Composite Data Mining Approach

We use a composite model based on the different contribution factors. The composite model combines two data mining approaches (Figure 1). First, we apply a Naive Bayesian classifier (as used by spam filters) [3] on the message content (subject and body) to determine how relevant an email is to a developer. Second, we use *Thread*, *Social* and *Time dimension* with *the output of the Naive Bayesian classifier* as the input to a Decision Tree classifier.

Before discussing in more detail the two classifiers, we provide an example of a Decision Tree that is produced from our case studies (Figure 2). The Decision Tree is for “*Thomas Lockhart*”, one of the main PostgreSQL developers. The Decision Tree indicates that *Thomas Lockhart* will not contribute if there is more than one message in the thread (thread_length $>$ 1), the score from the Naive Bayesian classifier is $>$ 0.9, his contribution activity in the last month is

TABLE I
DIFFERENT CONTRIBUTION FACTORS USED IN OUR APPROACH.

Attribute Name	Dimension	Type	Explanation	Rationale
thread_length	Thread	Numeric	Number of messages that have been posted in the thread, before the contributing developer decides to contribute to the thread.	Long threads might decrease the interest of a developer in a thread, but also increase the probability that everything has been said already in the thread.
word_count	Thread	Numeric	Number of words in the threads that the contributing developer has to read before posting a reply to the thread.	Long messages might decrease the interest of a developer in a thread or actually, but also increase the probability that everything has been said already in the thread.
known_starter	Social	Boolean	Check if the developer who is going to contribute to the thread knows the starter of the thread.	Developers might prefer to take part in threads that were started by someone they know.
known_poster	Social	Boolean	Check if the developer who is going to contribute to the thread knows the developer who posted the last reply in the thread.	Developers might prefer to take part in threads with the last message coming from someone they know.
contribution_activity	Social	Numeric	The number of threads a developer contributed to in the last month.	Inactive developers are less likely to contribute to a thread (e.g., they might be on vacation or have quit the project).
msg_time	Time	Numeric	The hour (0-23) when the last message in the thread was posted before the developer decides to post a reply.	A developer might prefer to contribute to the mailing list at set times during a day (e.g., working hours).
msg_day	Time	Nominal	The day of the week (Mon, Tue, Wed, Thu, Fri, Sat or Sun) when the last message was posted before the developer decides to contribute.	A developer might prefer to contribute to the mailing list on a particular day of the week (e.g., weekends or a working day).
msg_month	Time	Nominal	The month (Jan, Feb, March, April, May, June, July, Aug, Sep, Oct, Nov or Dec) when the last message was posted before the developer decides to contribute.	The developer contribution activity may change depending on the month (e.g., Christmas period).
subject	Content	String	The original subject of a thread.	The subject of a thread is the first thing the developer considers before deciding to contribute to the thread.
body	Content	String	The aggregated body text of all the messages in the thread before the developer decides to post a reply.	The body contains the actual thread content and is basically the reason why people send emails in the first place.

low (contribution_activity is ≤ 200) and the number of words in the message is ≤ 7175, or he does not know the poster.

The **Naive Bayesian classifier** works as follows. In the training phase, the Naive Bayesian classifier takes the content dimension from the training corpus and splits it into two corpora. One corpus contains the subject and the content of the threads that the developer contributed to (equivalent to non-spam messages). The other corpus contains the content of the threads that the developer did not contribute to (equivalent to spam messages). In the training phase, each message is divided into tokens (word) and in each corpus we count the occurrences of each token. We use these counts to determine the probability of each token to be an indicator of spam threads (i.e. thread that developers should not contribute to). Finally, we combine the highest 15 probabilities [4]) together into one probability, which gives a score (probability) that a developer will contribute to a given thread. The closer the score is to 1, the higher the probability that the developer will not contribute to the thread.

The **Decision Tree classifier** takes the score from the Naive Bayesian classifier algorithm as input instead of the message content dimension, together with the other factors discussed in the previous subsection. We choose to use a Decision Tree classifier as a machine learning algorithm, since a Decision Tree classifier offers an explainable model. Such a model explicitly shows the major factors that affect a developer's decision to contribute, while many of the other machine learning approaches produce black box models that do not explain their classification decisions. We choose to replace the message content by the output score of the Naive Bayesian classifier, because Decision Trees do not support string factors.

We used the C4.5 algorithm [5] to create our Decision Tree. This algorithm starts with an empty tree. Then, at each level, the algorithm calculates the information gain using the value of each of the factors listed in Table I. The information gain measures the improvement in classification accuracy for the training data when using a particular factor at that specific level. The factor with the highest information gain is added as a decision node. This process is repeated at each level until the number of training instances classified in the lowest level (i.e., leaves) reaches a specified minimum [5].

C. Evaluating the Accuracy of our approach

We use a confusion matrix to evaluate the accuracy of our approach. The confusion matrix contains the classification decisions made by the Decision Tree classifier versus the real classifications in the mailing list data (Contribute and Not-contribute). Table II shows an example for a confusion matrix. We measure the following three misclassification rates:

1) **Contribute misclassification:** This captures the percentage of times when a developer did contribute, but the classifier determined that the developer would not contribute. This misclassification rate is calculated as: $b/(a+b)$.
2) **Not-contribute misclassification:** This captures the percentage of times when a developer did not contribute to a thread but the classifier determined that the developer would contribute. This misclassification rate is calculated as: $c/(c+d)$.
3) **Overall misclassification:** This captures the percentage of wrong classifications made by the classifier. It is calculated as: $(b+c)/(a+b+c+d)$.

TABLE II
THE CONFUSION MATRIX FOR OUR MODELS.

	Classified as	
True Class	Contribute	Not contribute
Contribute	a	b
Not contribute	c	d

If the *Contribute* misclassification rate is high, then developers are likely to miss contributing to threads that need their input. If the *Not-contribute* misclassification rate is high, then developers might waste time focusing on irrelevant threads. Model's **accuracy** is (1 - overall misclassification rate).

Ideally, we seek a model with minimal misclassification rates. Our top priority is to minimize the *Contribute* misclassification rate. We believe that it might be better to read additional discussions than to miss a thread that requires the contribution of a developer.

We use tenfold cross-validation to estimate the accuracy of our model. A tenfold cross-validation divides the threads into two parts: testing corpus–containing one tenth of the contribution factors, and training corpus–containing the rest of the contribution factors. The training corpus is used to build the classification model, while the testing corpus is used to test the accuracy of the model. This process is repeated ten times, each time shifting the fold that is used for testing. Tenfold cross-validation gives more accurate results than other validation approaches, such as Holdout and Bootstrap approaches [6].

III. CASE STUDY SETUP

This section presents our setup approach for extracting the messages from the mailing list archive, reconstructing the discussion threads and data preparation.

A. Data Collection

We process each message in a mailing list repository using a semi-automated approach similar to [7]. We remove attachments, duplicate messages, convert HTML emails to plain text, and extract the email header information (such as From and Date). Then, we identify and merge multiple email addresses that belong to the same person. Such steps are needed to ensure the correctness of the calculation of the factors listed in Table I. For example, Alvaro Herrera, one of the most active developers in the PostgreSQL mailing list, uses six different email addresses. We must unify these addresses as a single person to ensure the correctness of our analysis.

B. Reconstruction of Discussion Threads

Each discussion thread consists of several email messages. However, email messages are stored in mailing list archives based on the posting time and date of the message. We must reconstruct the discussion threads by linking these email messages together. We use three heuristics to reconstruct threads. Each heuristic adds new messages to threads generated by the previous heuristic and creates new threads. The heuristics are:

The "InReplyTo" heuristic. This heuristic uses the in-reply-to field of a message to find the ID of earlier messages in a thread.

The "References" heuristic. This heuristic uses the references field of a message to find the ID of earlier messages in a thread. This field contains all the IDs of the previous messages sent to a thread. We use this heuristic on the remaining messages that were not classified using the InReplyTo heuristic.

The "Subject" heuristic. In many cases, email clients fail to generate message-ids. Also, old mailing list did not use message-ids before 1998. In this case, we can still deduce the thread of a message from its subject field. To avoid merging messages from two separate discussions that happen to have the same subject, we use a sliding time window of 6 months. We used this heuristic on the remaining unthreaded messages.

C. Data Preparation

To prepare our data for the experiment, we create a training model for each developer. We take all the messages in a discussion thread up until the contribution of a developer, and we mark the thread as a contribute thread for that developer. If the developer did not contribute to a given thread we take all the messages posted in that thread and mark the thread as a not-contribute thread.

For instance, if we are building a model for the developer "Andrew Dunstan" and we have a thread with 10 postings and "Andrew Dunstan" posted the fifth post, we would generate one contribute that contains all the aggregate text in the body of the messages until the fifth post and discard the other five posts (for this developer). This approach in preparing our data ensures that our models are more realistic, since our approach calculates the contribution activity until a developer contributes to a thread.

IV. CASE STUDY

We mine the developer mailing lists of three open source systems (Apache, PostgreSQL and Python) from different domains. The aim of this case study is to build a model that helps to understand the important factors that motivate developers to contribute to a given thread. Table III summarizes the details of the studied software systems. A historical archive of all discussions for each mailing list is publicly available online as monthly mbox files.

TABLE III
SUMMARY OF THE STUDIED SYSTEMS.

Project	Apache	PostgreSQL	Python
Domain	Web Server	DBMS	Interpreter
# of Messages	121,288	162,741	93,919
# of Threads	18,838	18,945	10,671
# of Contributors	3,137	4,996	2,848
Start date	March 1995	Jan. 1997	April 1999
End date	Dec. 2009	Sept. 2008	Dec. 2009
% of messages posted by top 10 developers	43%	36%	40%

We choose to study the performance of our models for the top ten most active developers. We feel that such developers, who are flooded with emails, will benefit the most from our approach over casual developers on the mailing lists. Table IV shows the activity of the top 10 developers who dominate the mailing list discussions for the three projects.

Due to space limitations, we only discuss in detail the results of the PostgreSQL project and we briefly present the results of the other two projects. We choose to explore and discuss PostgreSQL in detail, since it has the largest number of threads and messages among the studied projects. Moreover, Table IV shows that the PostgreSQL project has the largest variation in the number of threads the top 10 developers contributed to. We want to examine closely the performance of our approach under such high variation.

Q1. Can we build a high accuracy model of developer contribution to a thread?

We use our methodology from section 2 to build a composite model based on all available mailing list data of each project. Such a model will enable us to understand the important factors that motivate developers to contribute to a given thread.

Table V shows the misclassification rates for the top ten developers for PostgreSQL. Lower misclassification rates are desired, especially for the **Contribute misclassification rate**. We find that the *Contribute misclassification rate* for Tom Lane is low (18%), while for the other developers the *Contribute misclassification rates* are high (29% to 51%). On the other hand, the *Not-contribute misclassification rate* for all the developers is between 0% and 2%, except for Tome Lane (13%) and Bruce Momjian (5%). The *Overall misclassification rate* for all the developers ranges from 2% to 16%. These results are acceptable if we care about the overall performance, but we are interested more in **the Contribute class**. It is important that developers do not miss contributing to an important thread, but it is acceptable to read more threads.

The problem is Decision Trees tend to bias their classification to the majority class, especially when that class is much larger than the other classes, as in our case [8]. Table VI shows the ratio between the "*Contribute*" class and "*Not-Contribute*" class for each developer. If the contribute ratio is near 1, then

TABLE V
MISCLASSIFICATION RATES (IN PERCENTAGES).

Name	Contribute	Not contribute	Overall
Tom Lane	18	13	16
Bruce Momjian	31	5	14
Peter Eisentraut	35	2	6
Thomas Lockhart	41	1	4
Andrew Dunstan	35	1	4
Alvaro Herrera	51	0	4
The Hermit Hacker	38	0	3
Christopher Kings	34	1	3
Hannu Krosing	37	1	2
Jan Wieck	29	0	2

TABLE VI
THE CONTRIBUTION RATIO FOR POSTGRESQL PROJECT.

Name	**contribution ratio**
Tom Lane	1.18
Bruce Momjian	0.52
Peter Eisentraut	0.13
Thomas Lockhart	0.09
Andrew Dunstan	0.09
Alvaro Herrera	0.08
The Hermit Hacker	0.08
Christopher Kings	0.06
Hannu Krosing	0.05
Jan Wieck	0.05

the size of the "*Contribute*" and "*Not-contribute*" class are nearly equal. But if the contribute ratio is less than 1, then the "*Not-contribute*" class dominates the "*Contribute*" class. Table VI shows that the "*Not-contribute*" class clearly dominates the "*Contribute*" class as the ratios approach zero, except for Tom Lane and Bruce Momjian. In short, the Decision Tree simply learns the majority without trying to learn any factors from the minority training data. This observation appears in many real-world applications (e.g., in vision recognition [9], bioinformatics [10], credit card fraud detection [11], cancer detection [12], bug prediction [13] and bug triage [14]).

To tackle the problem of highly imbalanced classes, we must increase the minority class (Contribute class) to improve the *Contribute misclassification rate*. Re-balancing the training data is a frequently used technique to address this problem. There are two approaches for re-balancing the data:

1) **By re-weighting the minority class.** Re-weighting the minority class by assigning a higher weight to the minority class ensures that the Decision Tree would consider the minority class more prominently.
2) **By re-sampling the data.** Re-sampling the data can be done by under-sampling, over-sampling or both [15]. Estabrooks and Japkowicz [15] note that the best approach is to perform a combined under-sampling and over-sampling, as under-sampling alone discards useful data and over-sampling leads to over-fitted models.

We rebuild our models using re-weighting and re-sampling approaches. We use the AdaBoost algorithm [16] which is part

TABLE IV
THE TOP 10 MAILING LIST DEVELOPERS ORDERED BY THE NUMBER OF THREADS THEY CONTRIBUTED TO.

(a) Apache Project

Name	contributed to
Ken Coar	3,267
Jim Jagielski	3,212
William Rowe	3,167
Rob Hartill	3,112
Dean Gaudet	2,998
Marc Slemko	2,447
Brian Behlendorf	2,392
Ben Lauri	2,333
Jeff Trawick	2,149
Randy Terbush	1,972

(b) PostgreSQL Project

Name	contributed to
Tom Lane	10,184
Bruce Momjian	6,423
Peter Eisentraut	2,180
Thomas Lockhart	1,580
Andrew Dunstan	1,521
Alvaro Herrera	1,403
The Hermit Hacker	1,388
Christopher Kings	1,102
Hannu Krosing	957
Jan Wieck	860

(c) Python Project

Name	contributed to
Guido van Rossum	3,597
Martin Von Loewis	2,342
Tim Peters	1,974
Skip Montanaro	1,303
Barry Warsaw	1,260
Greg Ewing	1,126
M. Lemburg	1,079
Nick Coghlan	1,001
Aahz Maruch	895
Fredrik Lundh	893

TABLE VII
MISCLASSIFICATION RATES (IN PERCENTAGES) AFTER RE-SAMPLING THE DATASET. PERFORMANCE DIFFERENCES RELATIVE TO TABLE V ARE SHOWN BETWEEN PARENTHESES.

Name	Contribute	Not contribute	Overall
Tom Lane	19(+ 1)	15(+ 2)	17(+ 1)
Bruce Momjian	21(-10)	15(+10)	17(+ 3)
Peter Eisentraut	15(-20)	11(+ 9)	11(+ 5)
Thomas Lockhart	18(-23)	17(+16)	17(+13)
Andrew Dunstan	18(-17)	12(+11)	13(+ 9)
Alvaro Herrera	25(-26)	17(+17)	18(+14)
The Hermit Hacker	18(-20)	14(+14)	15(+12)
Christopher Kings	14(-20)	8(+ 7)	8(+ 5)
Hannu Krosing	20(-17)	23(+22)	21(+19)
Jan Wieck	19(-10)	7(+ 7)	7(+ 5)

of the WEKA machine learning framework [17] to perform a combined over- and under-sampling of our training data. We find that a re-weighting or re-sampling approach outperforms the original non-balanced approach. However, re-sampling is better than re-weighting by 1 to 4%, so we will use the re-sampling approach throughout the rest of the paper.

Table VII shows the misclassification rates using the re-sampling approach. The *Contribute misclassification rates* for all the developers, except for Tom Lane, have dramatically improved by as much as 26%. However, this improvement leads to a 7% to 17% increase in the *Not-contribute misclassification rate*. However, the *Overall and Contribute misclassification rates* for Tom Lane change by 1% and the *Not-contribute misclassification rate* changes only by 2%. This small decrease in the performance is expected because the contribute ratio for Tom Lane was reasonably balanced before re-sampling.

Our new models ensure to build a model with high accuracy that explains the contribution behavior for the developers. Our composite model ensures that a developer misses contributing to a smaller number of relevant discussion threads, while she will have to look through a larger number of discussion threads that do not require their contribution.

Our findings hold across all three studied projects (Apache, PostgreSQL and Python). Table VIII shows the average misclassification rates for the three projects. The misclassification rates are consistent across the three projects with average *Overall misclassification rate* between 11% and 14.4%.

TABLE VIII
THE AVERAGE MISCLASSIFICATION RATES (IN PERCENTAGES) AMONG THE TOP 10 DEVELOPER FOR EACH PROJECT AFTER RE-SAMPLING THE DATASET.

Project	Contribute	Not contribute	Overall
Apache	12.7	12.5	12.6
PostgreSQL	18.7	13.9	14.4
Python	19.3	9.8	11.0

We can build a developer contribution model with a high accuracy of 89–895% using data re-sampling.

Q2. What are the most important factors that influence a developer to contribute to a thread?

We now study which factors have the most influence on the decision of a developer to contribute. This study helps us to understand the contribution behavior of developers. In order to study the most important factors, we examine the top factors in the ten Decision Trees created by the Tenfold cross validation for each developer. The most important factor is the root node of a Decision Tree. The factors gradually become less important as we go down the tree. If the thread_length factor would appear as the root node in each of the ten trees created in each fold, then we would record that thread_length appears ten times at Level 0. This would mean that thread_length is the most important factor. This analysis is called **Top Node analysis** [18], [19].

Due to space constraints, Table IX only shows the Top Node analysis results for the most active developer (Tom Lane), a middle activity developer (Alvaro Herrera), and the least active developer (Jan Wieck) from the top ten most active developers. We also show the average Top Node analysis results across the top ten developers. Looking at the results for the three developers, we note that each developer has different contribution behavior. For Tom Lane, our model suggests that the length of the discussion is the most important factor, as it appears in the root of every tree. Alvaro Herrera's contribution activity is considered to be the most important factor, since Alvaro's contribution activity appears in eight out of the ten

TABLE IX
TOP NODES IN DECISION TREES BUILT FOR TABLE VII.

	Tom Lane		Alvaro Herrera		Jan Wieck		Average	
Level	#	Attribute	#	Attribute	#	Attribute	#	Attribute
0	10	thread_length	8	contribution_activity	10	Naive_Bayesian_score	4	Naive_Bayesian_score
			2	thread_length			3	contribution_activity
							3	thread_length
1	10	Naive_Bayesian_score	6	Naive_Bayesian_score	8	thread_length	5	Naive_Bayesian_score
			3	thread_length	2	contribution_activity	4	thread_length
			1	contribution_activity			1	word_count
							1	contribution_activity
2	10	words_count	5	thread_length	5	contribution_activity	3	words_count
			2	Naive_Bayesian_score	2	word_count	2	Naive_Bayesian_score
			2	words_count	2	thread_length	2	contribution_activity
			1	contribution_activity	1	msg_time	2	thread_length
							1	msg_time

TABLE X
THE AVERAGE TOP NODES IN DECISION TREES FOR EACH PROJECT.

	Apache		PostgreSQL		Python	
Level	#	Attribute	#	Attribute	#	Attribute
0	5	Naive_Bayesian_score	4	Naive_Bayesian_score	7	Naive_Bayesian_score
	4	thread_length	3	thread_length	3	thread_length
	1	contribution_activity	3	contribution_activity		
1	3	Naive_Bayesian_score	5	Naive_Bayesian_score	3	Naive_Bayesian_score
	2	thread_length	4	thread_length	2	thread_length
	2	word_count	1	word_count	2	contribution_activity
	2	contribution_activity	1	contribution_activity	1	word_count
2	2	words_count	3	words_count	1	words_count
	2	contribution_activity	2	Naive_Bayesian_score	1	contribution_activity
	1	Naive_Bayesian_score	2	contribution_activity	1	Naive_Bayesian_score
	1	thread_length	2	thread_length	1	thread_length
	1	known_starter	1	msg_time		

trees. For Jan Wieck, the message content (Naive Bayesian classifier score) seems to be very important in his contribution decision, since it appears in all his trees. For the average across the ten developers, we find that the message content comes first, followed by contribution activity and the length of the thread respectively.

We examine the top words that the Naive Bayesian classifier uses to define if Tom Lane and Jan Wieck are going to contribute to a thread. Through these words, we can gain insight into the behavior of developers and their areas of interest and expertise. For Tom Lane, the classifier uses the following words: "Linux", "baseline", "version", "package", "deadlock_timeout", "trace", "structure", and "debug". For Jan Wieck, the classifier uses the words: "format", "bug", "directory", "libpq", "reporting", "compile", "log_directory", "parser", and "testtable". The words for Tom match his leading role in the PostgreSQL project as the words show that he contributes to threads that talk about releases and overall structuring issues. For Jan, the words highlight his areas of expertise, which relate to the libpq library (this is the C application programmer's interface to PostgreSQL). Jan tends to contribute to threads that talk about various problems and bugs related to this library.

We also perform the Top Node analysis on the Apache and Python projects. The average top node analysis for each project is shown in Table X. Although there is no common contribution behavior for all developers, Table X shows that there are three main contribution factors that influence developers across all the studied projects: the Naive Bayesian score (developer's expertise), the message length (necessity to contribute) and the developer activity. The Top Node analysis shows that social dimension (inter-personal behavior) and time dimension (availability) do not influence the contribution behavior of the top 10 developers.

The content of the thread (Naive_Bayesian_score), the length of a thread (thread_length) and the contribution activity of a developer (contribution_activity) are the most important contribution factors.

V. DISCUSSION

In this section, we closely study the top two factors that influence the developers to contribute according to our models. The first factor is the message content, while the second factor is the length of the thread. We want to understand how these contribution factors affect our model compared to the other contribution factors.

A. Message content

In the previous section, we found that message content is one of the most important factors, as it appears either at

TABLE XI
MISCLASSIFICATION RATES (IN PERCENTAGES) AFTER APPLYING NAIVE BAYESIAN CLASSIFIER ONLY AND RE-SAMPLING THE DATASET. PERFORMANCE DIFFERENCES RELATIVE TO TABLE VII ARE SHOWN BETWEEN PARENTHESES.

Name	Contribute	Not contribute	Overall
Tom Lane	6(-13)	87(+72)	35(+18)
Bruce Momjian	28(+07)	57(+42)	50(+33)
Peter Eisentraut	69(+54)	17(+ 6)	22(+11)
Thomas Lockhart	91(+73)	3(-14)	25(+ 8)
Andrew Dunstan	80(+62)	7(- 5)	20(+ 7)
Alvaro Herrera	80(+55)	1(-16)	22(+ 4)
The Hermit Hacker	92(+74)	1(-13)	27(+12)
Christopher Kings	88(+74)	4(- 4)	6(- 2)
Hannu Krosing	94(+74)	7(-16)	10(-11)
Jan Wieck	100(+81)	0(- 7)	1(- 6)

TABLE XII
MISCLASSIFICATION RATES (IN PERCENTAGES) AFTER RE-SAMPLING THE DATASET WITHOUT USING MESSAGE CONTENT. PERFORMANCE DIFFERENCES RELATIVE TO TABLE VII ARE SHOWN BETWEEN PARENTHESES.

Name	Contribute	Not-contribute	Overall
Tom Lane	27(+ 8)	11(- 4)	20(+ 3)
Bruce Momjian	41(+20)	12(- 3)	22(+ 5)
Peter Eisentraut	32(+17)	33(+22)	32(+21)
Thomas Lockhart	19(+ 1)	26(+ 9)	26(+ 9)
Andrew Dunstan	21(+ 3)	26(+14)	25(+12)
Alvaro Herrera	23(- 2)	29(+12)	29(+11)
The Hermit Hacker	16(- 2)	21(+ 7)	21(+ 6)
Christopher Kings	24(+10)	28(+20)	28(+20)
Hannu Krosing	42(+22)	33(+10)	33(+12)
Jan Wieck	37(+18)	25(+18)	26(+19)

level 0 or level 1 in the Decision Trees. We want to compare our model with a model that only uses the Naive Bayesian classifier to find out if we actually need the other factors. To explore this question, we redo our experiment twice, once using only the Naive Bayesian classifier (Table XI), and once using the Decision Tree without the message content factor “Naive Bayesian score” (Table XII).

Table XI shows that using only a Naive Bayesian classifier dramatically increases one of the misclassification rates, either the not-contribute class as for Tom Lane and Bruce Momjian, or the contribute class for the other developers. Table XII shows that ignoring the message content leads to an increase in the *Overall misclassification rates* by 3% to 20%. The *Contribute misclassification rate* increases by 8% to 22%, except for Alvaro Herrera and The Hermit Hacker, for whom it decreases by 2%. The *Not-contribute misclassification rate* increased by 7% to 22%, except for Tom Lane and Bruce Momjian, for whom it decreased by 4% and 3% respectively.

Table XIII shows the average misclassification rate among the three projects using Naive Bayesian classifier only. The table shows that the *Contribute misclassification rate* for Apache and Python is lower than for PostgreSQL, but still they are higher than the *Contribute misclassification rate* using all contribution factors (Table VIII). The reason of having lower

TABLE XIII
THE AVERAGE MISCLASSIFICATION RATES (IN PERCENTAGES) AMONG THE TOP 10 DEVELOPER FOR EACH PROJECT AFTER APPLYING BAYESIAN CLASSIFIER ONLY AND RE-SAMPLING THE DATASET. PERFORMANCE DIFFERENCES RELATIVE TO TABLE VIII ARE SHOWN BETWEEN PARENTHESES.

Project	Contribute	Not contribute	Overall
Apache	31.7(+19.0)	11.8(-0.7)	12.5(-0.1)
PostgreSQL	72.8(+54.1)	18.4(+4.5)	21.8(+7.4)
Python	31.1(+11.8)	7.1(-2.7)	9.3(-1.7)

TABLE XIV
THE AVERAGE MISCLASSIFICATION RATES (IN PERCENTAGES) AMONG THE TOP 10 DEVELOPER FOR EACH PROJECT AFTER RE-SAMPLING THE DATASET WITHOUT USING MESSAGE CONTENT. PERFORMANCE DIFFERENCES RELATIVE TO TABLE VIII ARE SHOWN BETWEEN PARENTHESES.

Project	Contribute	Not contribute	Overall
Apache	21.3(+08.6)	23.6(+11.1)	23.3(+10.7)
PostgreSQL	28.2(+09.5)	24.4(+10.5)	26.2(+11.8)
Python	31.0(+11.7)	22.6(+12.8)	23.9(+12.9)

Contribute misclassification rate for Apache and Python, is that the Decision Trees for Apache and Python (some even exclusively) use the Naive Bayesian Score, whereas those for PostgreSQL use the other factors more. Table XIV shows the average misclassification rates for each project without using message content. The table shows that not using message content increases the *Overall misclassification rate* by 10.7% up to 12.9%. This means that using the other factors with the Naive Bayesian classifier reduces the misclassification rate.

B. Thread Length

The Top Node analysis shows that the length of a discussion (thread_length) is one of the most important factors for many of the developers. However, while examining a few of the Decision Trees, we note that some of the nodes state that the developer should contribute to a thread when the thread_length $\leq$ 1. Figure 2 shows an example of this in the tree generated for Thomas Lockhart. This pattern is not consistent for all developers though, as it does not hold for Christopher Kings and Hannu Krosing. This means that our model suggests that whenever there is a new thread posted and the developer is available online, then the developer should contribute to this new thread. This seems rather strange as developers are not available online 24 hours a day, seven days a week. However, by the time they get online, many emails have been answered already by other developers. Therefore, we do not expect each developer to reply to every single thread that is started.

To measure the effect of our composite model without using the thread_length factor, we redo our case study using the re-sampled training data, excluding the thread_length factor. Table XV shows that removing the thread_length factor improves the *Contribution misclassification rate* for some developers and increases it for others. The *Not-contribute and Overall misclassification rates* increase for all developers, except for the last three and Alvaro Herrera (which was expected since

TABLE XV
MISCLASSIFICATION RATES (IN PERCENTAGES) AFTER RE-SAMPLING THE DATASET WITHOUT USING THREAD_LENGTH. PERFORMANCE DIFFERENCES RELATIVE TO TABLE VII ARE SHOWN BETWEEN PARENTHESES.

Name	contribute	Not-contribute	Overall
Tom Lane	20(+1)	24(+ 9)	22(+ 5)
Bruce Momjian	18(-3)	30(+15)	26(+ 9)
Peter Eisentraut	16(+1)	13(+ 2)	13(+ 2)
Thomas Lockhart	22(+4)	24(+ 7)	24(+ 7)
Andrew Dunstan	17(-1)	22(+10)	22(+ 9)
Alvaro Herrera	21(-4)	14(- 3)	15(- 3)
The Hermit Hacker	26(+8)	22(+ 8)	23(+ 8)
Christopher Kings	15(+1)	7(- 1)	7(- 1)
Hannu Krosing	20(+0)	9(-14)	9(-12)
Jan Wieck	19(+0)	7(+ 0)	7(+ 0)

TABLE XVI
THE AVERAGE MISCLASSIFICATION RATES (IN PERCENTAGES) AMONG THE TOP 10 DEVELOPERS FOR EACH PROJECT AFTER RE-SAMPLING THE DATASET WITHOUT USING THREAD_LENGTH. PERFORMANCE DIFFERENCES RELATIVE TO TABLE VIII ARE SHOWN BETWEEN PARENTHESES.

Project	Contribute	Not contribute	Overall
Apache	12.3(-0.4)	17.3(+4.8)	16.9(+4.3)
PostgreSQL	19.4(+0.7)	17.2(+3.3)	16.8(+2.4)
Python	20.0(+0.7)	12.9(+3.1)	13.5(+2.5)

our Top Node analysis shows that thread_length is not an important factor for them).

The average misclassification rates for the three projects without using thread length are shown in Table XVI. The table shows that removing thread length increases the *Overall and Not-contribute misclassification rates*. Also, thread length has a small effect on the *Contribute misclassification rate* as the performance changes by less than 1% than the original model (Table VIII). This means that thread length only slightly improves our composite model, although thread length appears at the top levels in the Decision Trees.

C. Developer Privacy

When writing this paper, we struggled with a dilemma about using the real names of the developers to safeguard the privacy of the developers. Eventually, we decided not to anonymize their names for two reasons.

- **Publicly Accessible Data**: We use developer mailing lists of three open source systems, in which data is publicly available. Hence, it is hard to hide developer names. One way to decipher developer names is by ordering the developer names by the number of threads they contributed to and compare these names with Table IV.
- **Case Study Replication**: Researchers normally replicate a case study to compare it with a new approach. Some years ago, a researcher had to replicate our case study [20] and contacted us to decipher the anonymized names.

Eventually, we decided to use the real names of the developers, to make it easier for researchers to replicate our case study and improve our results.

VI. THREATS TO VALIDITY

Our case study was performed using the development mailing list of three open source projects with different domain and size. Additional case studies on commercial projects are needed to verify the generality of our findings.

In our study, we use the whole history of the mailing list. Building models per year may result in different findings. In future work, we plan to study the developer contribution characteristics on yearly bases.

We use Decision Trees to build our models, but other techniques such as Support Vector Machines (SVM) should be studied and compared. Also, our model uses a small set of contribution factors, but they all perform well. However, additional factors should be explored, because they might improve the performance of our model. Prior work by Bird [21] uses a subset of our factors.

Marking a thread for a given developer as not to contribute to it, does not mean that the developer did not read the thread, but rather that it means either he does not have the correct information to contribute to this thread or someone already posted the correct contribution. Also, when our model suggests not to contribute to a given thread, this does not mean that the developer should not read the message.

VII. RELATED WORK

Our survey of related work focuses on two categories: work that mines mailing lists and work that mines software archives to cope with overload.

Using Mailing List Archives: Previous work uses mailing lists to study the social structure of developers. Bird *et al.* [22] created a developer social network and used it to study the evolution of sub-communities within large projects. Weissgerber *et al.* [23] used mailing lists to study the likelihood of a patch getting accepted and Rigby *et al.* [24] used mailing lists to study the code review process. We determine the contribution of developers to developer mailing list threads.

In addition, several studies used mailing lists to study developer morale, work times and the code review process. Rigby and Hassan [20] performed a psychometric study to identify the personality types of open-source software developers and to gain insight into the phenomena of pre- and post-release optimism. Tsunoda *et al.* [25] observed that, every year, an increasing number of commit messages are being sent during overtime periods. Our approach studies the factors that motivate developers to contribute to a given thread.

Other work used mailing lists to identify architectural changes [26], to accurately identify actors [27] and to study the time it takes for developers to be invited into the core group of a project [28]. Bacchelli et. al. [29] create a benchmark that evaluates the linking between source code and e-mails. The work closest to ours is the work done by Bird [21]. He applies neural networks to predict which emails a developer would be interested in. Our work differs from Bird's work in that Bird builds a single model for all the participants of a mailing list, while we build personalized models for each participant that has a better performance and presents an explainable model of the contribution behavior of each developer.

Using Software Archives to cope with overload: Vary researchers analyze software archives to assist overloaded people. Anvik *et al.* [14] applied a supervised machine learning algorithm on data mined from bug repositories to assist in the assignment of a bug report to a developer with the appropriate expertise. This study helps overloaded managers assign bugs to the right developer. Zimmerman *et al.* [30] created ROSE, a tool that uses source code repositories to recommend related files that may need to be co-changed. Ying *et al.* [31] created rule associations for files using information contained in the source code repository. Using these rules, Ying *et al.* assist in the identification of related changes that occur in the future. Our composite model identifies for overloaded developers which mailing list threads need their contribution.

VIII. Conclusions

We build a composite model that explores the contribution behavior for the top ten developers in mailing lists based on our factors in 4 dimensions. We applied our composite model on three open source mailing lists (Apache, PostgreSQL and Python). Our composite model shows that the contribution behavior varies between the developers, yet the models are intuitive and simple to follow. The most important contribution factors are the message content, developer contribution activity in the last month and the length of the thread. We are currently exploring whether our composite model could be used by the developers in practice to support them to identify which threads need their contributions from the hundreds of emails they receive everyday.

References

[1] E. Shihab, Z. M. Jiang, and A. E. Hassan, "Studying the use of developer irc meetings in open source projects," in *Proc. of 25th IEEE Int. Conf. on Software Maintenance (ICSM)*, September 2009, pp. 147–156.

[2] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks in postgres," in *Proc. of the int. workshop on Mining software repositories (MSR)*, 2006, pp. 137–143.

[3] T. A. Meyer and B. Whateley, "Spambayes: Effective open-source, bayesian based, email classification system," in *Proc. of the First Conf. on Email and Anti-Spam (CEAS)*, 2004.

[4] P. Graham, "A plan for spam," http://paulgraham.com/spam.html, 2002, last accessed, March 2010.

[5] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[6] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1995, pp. 1137–1145.

[7] N. Bettenburg, E. Shihab, and A. E. Hassan, "An empirical study on the risks of using off-the-shelf techniques for processing mailing list data," in *Proc. of the 25th IEEE Int.Conf. on Software Maintenance (ICSM)*, 2009, pp. 539–542.

[8] R. Barandela, J. S. Sánchez, V. García, and F. J. Ferri, "Learning from imbalanced sets through resampling and weighting," in *Proc. of the First Iberian Conf. on Pattern Recongnition and Image Analysis (IbPRIA)*, Mallorca, Spain, June 2003, pp. 80–88.

[9] J. Sánchez, R. Barandela, A. Marqués, and R. Alejo, "Performance evaluation of prototype selection algorithms for nearest neighbor classification," in *proc. of the 14th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, 2001, pp. 44–50.

[10] R. Barandela, J. S. Sánchez, V. García, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition Research (JPRR)*, vol. 36, no. 3, pp. 849–851, 2003.

[11] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection," in *Con. on Knowledge Discovery and Data Mining (KDD)*, 1998, pp. 164–168.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Artificial Intelligence Research (JAIR)*, vol. 16, pp. 321–357, 2002.

[13] J. Shirabad, T. Lethbridge, and S. Matwin, "Supporting software maintenance by mining software update records," in *Proc. of the IEEE Int. Conf. on Software Maintenance (ICSM)*, 2001, pp. 22–31.

[14] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. of the 28th Int. Conf. on Software Eng. (ICSE)*, 2006, pp. 361–370.

[15] A. Estabrooks and N. Japkowicz, "A mixture-of-experts framework for learning from imbalanced data sets," in *Proc. of the 4th Int.Conf. on Advances in Intelligent Data Analysis (IDA)*, 2001, pp. 34–43.

[16] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Second European Conf. on Computational Learning Theory (EuroCOLT)*, 1995, pp. 23–37.

[17] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with java implementations," *Special Interest Group on Management Of Data (SIGMOD)*, vol. 31, no. 1, pp. 76–77, 2002.

[18] J. Sayyad Shirabad, Ph.D. dissertation.

[19] A. E. Hassan and K. Zhang, "Using decision trees to predict the certification result of a build," *proc. of the 21st Int. Conf. on Automated Software Eng. (ASE)*, pp. 189–198, 2006.

[20] P. C. Rigby and A. E. Hassan, "What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List," in *Proc. of the Fourth Int. Workshop on Mining Software Repositories (MSR)*, 2007, pp. 23–31.

[21] C. Bird, "Predicting email response using mined data," http://wwwcsif.cs.ucdavis.edu/~bird/papers/mlpaper.pdf, last accessed, March 2010.

[22] C. Bird, D. Pattison, R. D'Souza, V. Folkiv, and P. Devanbu, "Latent Social Structure in Open Source Projects," in *Proc. of the 2008 ACM SIGSOFT symposium on the Foundations of Software Eng. (FSE)*, 2008, pp. 24–35.

[23] P. Weissgerber, D. Neu, and S. Diehl, "Small patches get in!" in *Proc. of the Int. Working Conf. on Mining Software Repositories (MSR)*. New York, NY, USA: ACM, 2008, pp. 67–76.

[24] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: A case study of the apache server," in *Proc. of the 30th Int. Conf. on Software Eng. (ICSE)*. New York, NY, USA: ACM, 2008, pp. 541–550.

[25] M. Tsunoda, A. Monden, T. Kakimoto, Y. Kamei, and K.-i. Matsumoto, "Analyzing oss developers' working time using mailing lists archives," in *Proc. of the Int. workshop on Mining Software Repositories (MSR)*, 2006, pp. 181–182.

[26] O. Baysal and A. J. Malton, "Correlating social interactions to release history during software evolution," in *Proc. of the Fourth Int. Workshop on Mining Software Repositories (MSR)*, 2007, p. 7.

[27] G. Robles and J. M. Gonzalez-Barahona, "Developer identification methods for integrated data from various sources," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[28] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu, "Open borders? immigration in open source projects." in *Proc. of the Fourth Int. Workshop on Mining Software Repositories (MSR)*, 2007, p. 6.

[29] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes, "Benchmarking lightweight techniques to link e-mails and source code," in *Working Conf. on Reverse Eng. (WCRE)*, 2009, pp. 205–214.

[30] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proc. of the 26th Int. Conf. on Software Eng. (ICSE)*, 2004, pp. 563–572.

[31] A. T. T. Ying, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng. (TSE)*, vol. 30, no. 9, pp. 574–586, 2004.