Noname manuscript No.
(will be inserted by the editor)

Studying Re-opened Bugs in Open Source Software

Emad Shihab[*] · **Akinori Ihara**[+] ·
Yasutaka Kamei[*] · **Walid M. Ibrahim**[*] ·
Masao Ohira[+] · **Bram Adams**[*] ·
Ahmed E. Hassan[*] · **Ken-ichi Matsumoto**[+]

Received: date / Accepted: date

Abstract *Context*: Bug fixing accounts for a large amount of the software maintenance resources. Generally, bugs are reported, fixed, verified and closed. However, in some cases bugs have to be re-opened. Re-opened bugs increase maintenance costs, degrade the overall user-perceived quality of the software and lead to unnecessary rework by busy practitioners.
Aim: In this paper, we study and predict re-opened bugs through a case study on three large open source projects – namely Eclipse, Apache and OpenOffice. We structure our study along 4 dimensions: 1) the work habits dimension (e.g., the weekday on which the bug was initially closed), 2) the bug report dimension (e.g., the component in which the bug was found) 3) the bug fix dimension (e.g., the amount of time it took to perform the initial fix) and 4) the team dimension (e.g., the experience of the bug fixer).
Method: We build decision trees using the aforementioned factors that aim to predict re-opened bugs. We perform top node analysis to determine which factors are the most important indicators of whether or not a bug will be re-opened.
Results: Our study shows that the comment text and last status of the bug when it is initially closed are the most important factors related to whether or not a bug will be re-opened. Based on these dimensions we create decision trees that predict whether a bug will be re-opened after its closure. Using a combination of our dimensions, we can build explainable prediction models that can achieve a precision between 52.1-78.6% and a recall in the range of 70.5-94.1% when predicting whether a bug will be re-opened.
Conclusions: We find that the factors that best indicate which bugs might be re-opened vary based on the project. The comment text is the most important factor for the Eclipse and OpenOffice projects, while the last status is the most important one for Apache. These

Software Analysis and Intelligence Lab (SAIL)[*]
Queen's University
Kingston, ON, Canada
Fax: +1-613-533-6513
E-mail: {emads, kamei, walid, bram, ahmed}@cs.queensu.ca

Graduate School of Information Science[+]
Nara Institute of Science and Technology
Takayama, Ikoma, Nara, JAPAN
E-mail: {akinori-i, masao, matumoto}@is.naist.jp

factors should be closely examined in order to reduce maintenance cost due to re-opened bugs.

Keywords Bug reports · Re-opened Bugs · Open Source Software

1 Introduction

Large software systems are becoming increasingly important in the daily lives of many people. A large portion of the cost of these software systems is attributed to their maintenance. In fact, previous studies show that more than 90% of the software development cost is spent on maintenance and evolution activities [1].

A plethora of previous research addresses issues related to software bugs. For example, software defect prediction work uses various code, process, social structure, geographic distribution and organizational structure metrics to predict buggy software locations (e.g., files or directories) [2–8]. Other work focuses on predicting the time it takes to fix a bug [9–11].

This existing work typically treats all bugs equally, meaning, the existing work did not differentiate between re-opened and new bugs. Re-opened bugs are bugs that were closed by developers, but re-opened at a later time. Bugs can be re-opened for a variety of reasons. For example, a previous fix may not have been able to fully fix the reported bug. Or the developer responsible for fixing the bug was not able to reproduce the bug and might close the bug, which is later re-opened after further clarification.

Re-opened bugs take considerably longer to resolve. For example, in the Eclipse Platform 3.0 project, the average time it takes to resolve (i.e., from the time the bug is initially opened till it is fully closed) a re-opened bug is more than twice as much as a non-reopened bug (371.4 days for re-opened bugs vs. 149.3 days for non-reopened bugs). An increased bug resolution time consumes valuable time from the already-busy developers. For example, developers need to re-analyze the context of the bug and read previous discussions when a bug is re-opened. In addition, such re-opened degrade the overall user-perceived quality of the software and often lead to additional and unnecessary rework by the already-busy practitioners.

This paper presents an exploratory study to determine factors that indicate whether a bug will be re-opened. Knowing which factors are attributed to re-opened bugs prepares practitioners to think twice before closing a bug. For example, if it is determined that bugs logged with high severity are often re-opened, then practitioners can pay special attention (e.g., by performing more thorough reviews) to such bugs and their fixes.

We combine data extracted from the bug and source control repositories of the Eclipse, Apache and OpenOffice open source projects to extract 24 factors that are grouped into four different dimensions:

1. **Work habits dimension**: which is used to gauge whether the work habits of the software practitioners initially closing the bug affect its likelihood of being re-opened. The main reason for studying this dimension is to possibly provide some insights about process changes (e.g., avoiding closing bugs during specific times) that might lead to a reduction in bug re-openings
2. **Bug report dimension**: which is used to examine whether information in the bug report can be used to determine the likelihood of bug re-opening. The main reason for examining this dimension is to see whether information contained in the bug report can be

used to hint a higher risk of a bug being re-opened in the future. Practitioners can then be warned about such data in order to reduce bug re-openings.

3. **Bug fix dimension**: which is used to examine whether the fix made to address a bug can be used to determine the likelihood of a bug being re-opened. The reason for studying this dimension is to examine whether certain factors related to the bug fix increase the likelihood of it being re-opened later. Insights about issues that might increase the likelihood of a bug being re-opened are helpful to practitioners so they can know what to avoid when addressing bugs.
4. **Team dimension**: which is used to determine whether the personnel involved with a bug can be used to determine the likelihood of a bug being re-opened. The reason for using this dimension is to examine whether certain personnel (e.g., more experienced personnel) should avoid or be encouraged to address bugs, in order to reduce bug re-openings.

To perform our analysis, we build decision trees and perform a Top Node analysis [12, 13] to identify the most important factors in building these decision trees. Furthermore, we use the extracted factors to predict whether or not a closed bug will be re-opened in the future. In particular, we aim to answer the following research questions:

Q1. Which of the extracted factors indicate, with high probability, that a bug will be re-opened?
The factors that best indicate re-opened bugs vary based on the project. The comment text is the most important factor for the Eclipse and OpenOffice projects, while the last status is the most important one for Apache.

Q2. Can we accurately predict whether a bug will be re-opened using the extracted factors?
We use 24 different factors to build accurate prediction models that predict whether or not a bug will be re-opened. Our models can correctly predict whether a bug will be re-opened with precision between 52.1-78.6% and recall between 70.5-94.1%.

This paper extends an earlier conference paper [41], in which we study and predict re-opened bugs on the Eclipse project. In this paper, we extend our previous work by conducting our study on two additional Open Source Systems (OSS), the Apache HTTP server and OpenOffice. Doing so reduces the threat to external validity and improves the generalizability of our findings since the three systems come from different domains (i.e., Integrated Development Environment (IDE) vs. Web Server vs. Productivity Suite) and are written in different programming languages (i.e., C/C++ vs. Java). In addition, we compare our findings for the three projects and provide insight about the way in which the important factors impact the likelihood of a bug being re-opened.

The rest of the paper is organized as follows. Section 2 describes the life cycle of a bug. Section 3 presents the methodology of our study. We detail our data processing steps in Section 4. The case study results are presented in Section 5. We compare the prediction results using different algorithms in Section 6. The threats to validity and related work are presented in Sections 8 and 9, respectively. Section 10 concludes the paper.

2 The Bug Life Cycle

Bug tracking systems, such as Bugzilla [14], are commonly used to manage and facilitate the bug resolution process. These bug tracking systems record various characteristics about

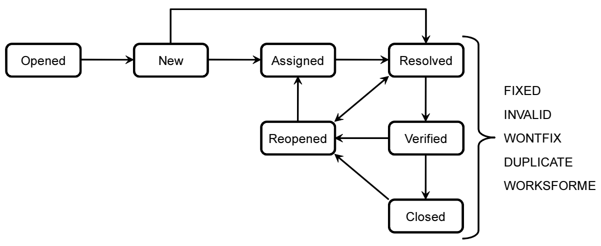

Fig. 1 Bug resolution process [15]

reported bugs, such as the time the bug was reported, the component the bug was found in and any discussions related to the bug. The information stored in bug tracking systems is leveraged by many researchers to investigate different phenomena (e.g., to study the time it takes to resolve bugs [18, 35]).

The life cycle of a bug can be extracted from the information stored in the bug tracking systems. We can track the different states that bugs have gone through and reconstruct their life cycles based on these states. For example, when bugs are initially logged, they are confirmed and labeled as new bugs. Then, they are triaged and assigned to developers to be fixed. After a developer fixes the bug, the fix is verified and the bug closed.

A diagram representing the majority of the states bugs go through is shown in Figure 1. When developers, testers or users experience a bug, they log/submit a bug report in the bug tracking system. The bug is then set to the *Opened* state. Next, the bug is triaged to determine whether it is a real bug and whether it is worth fixing. After the triage process, the bug is accepted and its state is updated to *New*. It then gets assigned to a developer who will be responsible to fix it (i.e., its state is *Assigned*). If a bug is known to a developer beforehand[1], it is assigned to that developer who implements the fix and the bug goes directly from the *New* state to the *Resolved_FIXED* state. More typically, bugs are assigned to a developer (i.e., go to the *Assigned* state), who implements a fix for the bug and its state is transitioned into *Resolved_FIXED*. In certain cases, a bug is not fixed by the developers because it is identified as being invalid (i.e., state *Resolved_INVALID*), a decision was made to not fix the bug (i.e., state *Resolved_WONTFIX*), it is identified as a duplicate of another bug (i.e., state *Resolved_DUPLICATE*) or the bug is not reproducible by the developer (i.e., state *Resolved_WORKSFORME*). Once the bug is resolved, it is verified by another developer or tester (state *Verified_FIXED*) and finally closed (state *Closed*).

However, in certain cases, bugs are re-opened after their closure. This can be due to many reasons. For example, a bug might have been incorrectly fixed and resurfaces. Another reason might be that the bug was closed as being a duplicate and later re-opened because it was not actually a duplicate.

In general, re-opened bugs are not desired by software practitioners because they degrade the overall user-perceived quality of the software and often lead to additional and unnecessary rework by the already-busy practitioners. Therefore, in this paper we set out to investigate which factors best predict re-opened bugs. Then, we use these factors to build accurate prediction models to predict re-opened bugs.

[1] For example, in some cases developers discover a bug and know how to fix it, however they create a bug report and assign it to themselves for book-keeping purposes.

Table 1 Factors Considered in Our Study

Dim	Factor	Type	Explanation	Rationale
Work habits	Time	Nominal	Time in hours (Morning, Afternoon, Evening, Night) when the bug was first closed.	Bugs closed at certain times in the day (e.g., late afternoons) are more/less likely to be re-opened.
	Weekday	Nominal	Day of the week (e.g., Mon or Tue) when the bug was first closed.	Bugs closed on specific days of the week (e.g., Fridays) are more/less likely to be re-opened.
	Month day	Numeric	Calendar day of the month (0-30) when the bug was first closed.	Bugs closed at specific periods like the beginning, mid or end of the month are more/less likely to be re-opened.
	Month	Numeric	Month of the year (0-11) when the bug was first closed.	Bugs closed in specific months (e.g., during holiday months like December) are more/less likely to be re-opened.
	Day of year	Numeric	Day of the year (1-365) when the bug was first closed.	Bugs closed in specific times of the year(e.g., later on in the year) are more/less likely to be re-opened.
Bug report	Component	Nominal	Component the bug was found in (e.g., UI, Debug or Search).	Certain components might be harder to fix; bugs found in these components are more/less likely to be re-opened.
	Platform	Nominal	Platform (e.g., Windows, MAC, UNIX) the bug was found in.	Certain platforms are harder to fix bugs for, and therefore, their bugs are more likely to be re-opened.
	Severity	Numeric	Severity of the reported bug. A high severity (i.e., 7) indicates a blocker bug and a low severity (i.e., 1) indicates an enhancement.	Bugs with high severity values are harder to fix and are more likely to be re-opened.
	Priority	Numeric	Priority of the reported bug. A low priority value (i.e., 1) indicates an important bug and a high priority value (i.e., 5) indicates a bug of low importance.	Bugs with low priority value (i.e., high importance) are likely to get more careful attention and have a smaller chance of being re-opened.
	Number in CC list	Numeric	Number of persons in the cc list of the logged bugs before the first re-open.	Bugs that have persons in the cc list are followed more closely, and hence, are more/less likely to be re-opened.
	Description size	Numeric	The number of words in the description of the bug.	Bugs that are not described well (i.e., have a short description) are more likely to be re-opened.
	Description text	Bayesian score	The text content of the bug description.	Words included in the bug description (e.g., complex or difficult) can indicate whether the bug is more likely to be re-opened.
	Number of comments	Numeric	The number of comments attached to a bug report before the first re-open.	The higher the number of comments, the more likely the bug is controversial. This might lead to a higher chance of it being re-opened.
	Comment size	Numeric	The number of words in all comments attached to the bug report before the first re-open.	The longer the comments are, the more the discussion about the bug and the more/less likely it will be re-opened.
	Comment text	Bayesian score	The text content of all the comments attached to the bug report before the first re-open.	The comment text attached to a bug report may indicate whether a bug will be re-opened.
	Priority changed	Boolean	States whether the priority of the bug was changed after the initial report before the first re-open.	Bugs that have their priorities increased are generally followed more closely and are less likely to be re-opened.
	Severity changed	Boolean	States whether the severity of the bug was changed after the initial report before the first re-open.	Bugs that have their severities increased are generally followed more closely and are less likely to be re-opened.
Bug fix	Time days	Numeric	The time it took to resolve a bug, measured in days. For re-opened bugs, we measure the time to perform the initial fix.	The time it takes to fix a bug is indicative of its complexity and hence the chance of finding a good fix.
	Last status	Nominal	The last status of the bug when it is closed for the first time.	When bugs are closed using certain statuses (e.g., Worksforme or duplicate), they are more/less likely to be re-opened.
	No. of files in fix	Numeric	The number of files edited to fix the bug for the first time.	Bugs that require larger fixes, indicated by an increase in the number of files that need to be edited, are more likely to be re-opened.
People	Reporter Name	String	Name of the bug reporter.	Bugs reported by specific individuals are more/less likely to be re-opened (e.g., some reporters may do a poor job reporting bugs).
	Fixer Name	String	Name of the initial bug fixer.	Bugs fixed by specific individuals are more/less likely to be re-opened (e.g., developers that work on complex functionality).
	Reporter Experience	Numeric	The number of bugs reported by the bug reporter before reporting this bug.	Experienced reporters know what information to provide in bug reports, therefore their bugs less likely to be re-opened.
	Fixer Experience	Numeric	The number of bug fixes the initial fixer performed before fixing this bug.	Experienced fixers are more familiar with the project, therefore their bugs are less likely to be re-opened.

3 Approach to Predict Re-opened Bugs

In this section, we describe the factors used to predict whether or not a bug will be re-opened. Then, we present decision trees and motivate their use in our study. Finally, we present the metrics used to evaluate the performance of the prediction models.

3.1 Dimensions Used to Predict if a Bug will be Re-opened

We use information stored in the bug tracking system, in combination with information from the source control repository of a project to derive various factors that we use to predict whether a bug will be re-opened.

Table 1 shows the extracted factors, the type of the factor (e.g., numeric or nominal), provides an explanation of the factor and the rationale for using each factor. We have a total of 24 factors that cover four different dimensions. We describe each dimension and its factors in more detail next.

Work habits dimension. Software developers are often overloaded with work. This increased workload affects the way these developers perform. For example, Sliwerski *et al.* [16] showed that code changes are more likely to introduce bugs if they were done on Fridays. Anbalagan *et al.* [17] showed that the time it takes to fix a bug is related to the day of the week when the bug was reported. Hassan and Zhang [12] used various work habit factors to predict the likelihood of a software build failure.

These prior findings motivate us to include the work habit dimension in our study on re-opened bugs. For example, developers might be inclined to close bugs quickly on a specific day of the week to reduce their work queue and focus on other tasks. These quick decisions may cause the bugs to be re-opened at a later date.

The work habit dimension consists of four different factors. The factors of the work habit dimension are listed in Table 1. The time factor was defined as a nominal variable that can be `morning` (7 AM to 12 Noon), `afternoon` (Noon to 5 PM), `evening` (5 PM to 12 midnight) or `night` (midnight to 7 AM), indicating the hours of the day that they bug was initially closed on.

Bug report dimension. When a bug is reported, the reporter of the bug is required to include information that describes the bug. This information is then used by the developers to understand and locate the bug. Several studies use that information to study the amount of time required to fix a bug [18]. For example, Panjar [9] showed that the severity of a bug has an effect on its lifetime. In addition, a study by Hooimeijer and Weimer [19] showed that the number of comments attached to a bug report affects the time it takes to fix it.

We believe that attributes included in a bug report can be leveraged to determine the likelihood of a bug being re-opened. For example, bugs with short or brief descriptions may need to be re-opened later because a developer may not be able to understand or reproduce them the first time around.

A total of 11 different factors make up the bug report dimension. They are listed in Table 1.

Bug fix dimension. Some bugs are harder to fix than others. In some cases, the initial fix to the bug may be insufficient (i.e., it did not fully fix the bug) and, therefore, the bug needs to be re-opened. We conjecture that more complicated bugs are more likely to be re-opened. There are several ways to measure the complexity of a bug fix. For example, if the bug fix requires many files to be changed, this might be an indicator of a rather complex bug [20].

The bug fix dimension uses factors to capture the complexity of the initial fix of a bug. Table 1 lists the three factors that measure the time it took to fix the bug, the status before the bug was re-opened and the number of files changed to fix the bug.

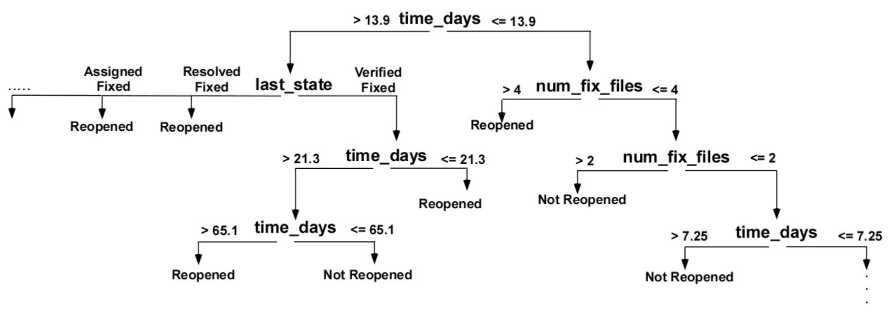

Fig. 2 Sample Decision Tree.

People dimension. In many cases, the people involved with the bug report or the bug fix are the reason that it is re-opened. Reporters may not include important information when reporting a bug, or they lack the experience (i.e., they have never reported a bug before). On the other hand, developers (or fixers) may lack the experience and/or technical expertise to fix or verify a bug, leading to the re-opening of the bug.

The people dimension, listed in Table 1, is made up of four factors that cover bug reporters, bug fixers and their experience.

The four dimensions and their factors listed in Table 1 are a sample of the factors that can be used to study why bugs are reopened. We plan (and encourage other researchers) to build on this set of dimensions to gain more insights into why bugs are re-opened.

3.2 Building Decision Tree-Based Predictive Models

To determine if a bug will be re-opened, we use the factors from the four aforementioned dimensions as input to a decision tree classifier. Then, the decision tree classifier predicts whether or not the bug will be re-opened.

We chose to use a decision tree classifier for this study since it offers an explainable model. This is very advantageous because we can use these models to understand what attributes affect whether or not a bug will be re-opened. In contrast, most other classifiers produce “black box” models that do not explain which attributes affect the predicted outcome.

To perform our analysis, we divide our data set into two sets: a training set and a test set. The training set is used to train the decision tree classifier. Then, we test the accuracy of the decision tree classifier using our test set.

The C4.5 algorithm [21] was used to build the decision tree. Using the training data, the algorithm starts with an empty tree and adds decision nodes or leafs at each level. The information gain using a particular attribute is calculated and the attribute with the highest information gain is chosen. Further analysis is done to determine the cut-off value at which to split the attribute. This process is repeated at each level until the number of instances classified at the lowest level reaches a specified minimum. Having a large minimum value means that the tree will be strict in creating nodes at the different levels. On the contrary, making this minimum value be small (e.g., 1) will cause many nodes to be added to the tree. To mitigate noise in our predictions and similar to previous studies [22], in our case study, we set this minimum node size to be 10.

To illustrate, we provide an example tree produced by the fix dimension, shown in Figure 2. The decision tree indicates that when the time_days variable (i.e., the number of days to fix the bug) is greater than 13.9 and the last status is Resolved Fixed, then the bug will be re-opened. On the other hand, if the time_days variable is less than or equal to 13.9 and the number of files in the fix is less than or equal to 4 but greater than 2, then the bug will not be re-opened. Such explainable models can be leveraged by practitioners to direct their attention to bugs that require closer review before they are closed.

3.3 Evaluating the Accuracy of Our Models

To evaluate the predictive power of the derived models, we use the classification results stored in a confusion matrix. Table 2 shows an example of a confusion matrix.

Table 2 Confusion Matrix

	True class	
Classified as	Re-open	Not Re-open
Re-open	TP	FP
Not Re-open	FN	TN

We follow the same approach used by Kim *et. al* [23], using the four possible outcomes for each bug. A bug can be classified as re-opened when it truly is re-opened (true positive, TP); it can be classified as re-opened when actually it is not re-opened (false positive, FP); it can be classified as not re-opened when it is actually re-opened (false negative, FN); or it can be classified as not re-opened and it truly is not re-opened (true negative, TN). Using the values stored in the confusion matrix, we calculate the widely used Accuracy, Precision, Recall and F-measure for each class (i.e., re-opened and not re-opened) to evaluate the performance of the predictive models.

The accuracy measures the number of correctly classified bugs (both the re-opened and the not re-opened) over the total number of bugs. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{1}$$

Since there are generally less re-opened bugs than not re-opened bugs, the accuracy measure may be misleading if a classifier performs well at predicting the majority class (i.e., not re-opened bugs). Therefore, to provide more insights, we measure the precision and recall for each class separately.

1. **Re-opened precision:** Measures the percentage of correctly classified re-opened bugs over all of the bugs classified as re-opened. It is calculated as P(re) $= \frac{TP}{TP+FP}$.
2. **Re-opened recall:** Measures the percentage of correctly classified re-opened bugs over all of the actually re-opened bugs. It is calculated as R(re) $= \frac{TP}{TP+FN}$.
3. **Not re-opened precision:** Measures the percentage of correctly classified, not re-opened bugs over all of the bugs classified as not re-opened. It is calculated as P(nre) $= \frac{TN}{TN+FN}$.
4. **Not re-opened recall:** Measures the percentage of correctly classified not re-opened bugs over all of the actually not re-opened bugs. It is calculated as R(nre) $= \frac{TN}{TN+FP}$.
5. **F-measure:** Is a composite measure that measures the weighted harmonic mean of precision and recall. For re-opened bugs it is measured as F-measure(re) $= \frac{2*P(re)*R(re)}{P(re)+R(re)}$ and for bugs that are not re-opened F-measure(nre) $= \frac{2*P(nre)*R(nre)}{P(nre)+R(nre)}$.

A precision value of 100% would indicate that every bug we classified as (not) re-opened was actually (not) re-opened. A recall value of 100% would indicate that every actual (not) re-opened bug was classified as (not) re-opened.

To estimate the accuracy of the model, we employ 10-fold cross validation [24]. In 10-fold cross validation, the data set is partitioned into 10 sets. Each of the 10 sets contains 1/10 of the total data. Each of the 10 sets is used once for validation (i.e., to test accuracy) and the remaining nine sets are used for training. We repeat the this 10-fold cross validation 10 times (i.e., we build 100 decision trees in total) and report the average.

4 Data Processing

To conduct our case study, we used three projects: Eclipse Platform 3.0, Apache HTTP Server and OpenOffice. The main reason we chose to study these three projects in our case study is because they are large and mature Open Source Software (OSS) projects that have a large user base and a rich development history. The projects also cover different domains (i.e., Integrated Development Environment (IDE) vs. Web Server vs. Productivity Suite) and are written in different programming languages (i.e., C/C++ vs. Java). We leveraged two data sources from each project, i.e., the bug database and the source code control (CVS) logs.

To extract data from the bug databases, we wrote a script that crawls and extracts bug report information from the project's online Bugzilla databases. The reports are then parsed and different factors are extracted and used in our study.

Most of the factors can be directly extracted from the bug report, however, in some cases we needed to combine the data in the bug report with data from the CVS logs. For example, one of our factors is the number of files that are changed to implement the bug fix. In most cases, we can use the files included in the submitted patch. However, sometimes the patch is not attached to the bug report. In this case, we search the CVS logs to determine the change that fixed the bug.

We used the J-REX [25] tool, an evolutionary code extractor for Java-based software systems, to perform the extraction of the CVS logs. The J-REX tool obtains a snapshot of the Eclipse CVS repository and groups changes into transactions using a sliding window approach [16]. The extracted logs contain the date on which the change was made, the author of the change, the comments by the author to describe the change and files that were part of the change. To map the bugs to the changes that fixed them, we used the approach proposed by Fischer *et al.* [43] and later used by Zimmermann *et al.* [2], which searches in the CVS commit comments for the bug IDs. To validate that the change is actually related to the bug, we make sure that the date of the change is on or prior to the close date of the bug. That said, there is no guarantee that the commits are bug fixes as they may be performing other types of changes to the code.

To use the bug reports in our study, we require that they be resolved and contain all of the factors we consider in our study. Table 3 shows the number of bug reports used from each project. To explain the data in Table 3, we use the Eclipse project as an example. We extracted a total of 18,312 bug reports. Of these 18,312 reports, only 3,903 bug reports were resolved (i.e., they were closed at least once). Of the resolved bug reports, 1,530 could be linked to source code changes and/or submitted patches. We use those 1,530 bug reports in our study. Of the 1,530 bug reports studied, 246 were re-opened and 1,284 were not.

For each bug report, we extract 24 different factors that cover four different dimensions, described in Table 1. Most of the factors were directly derived from the bug or code

Table 3 Bug Report Data Statistics

	Eclipse	**Apache HTTP**	**OpenOffice**
Total Extracted Bug Reports	18,312	32,680	106,015
Resolved Bug Reports	3,903	28,738	86,993
Bug Reports Linked to Code Changes or Patches	1,530	14,359	40,173
Re-opened Bug Reports	246	927	10,572
Not Re-opened Bug Reports	1,284	13,432	29,601

databases. However, two factors in the bug report dimension are text-based and required special processing. Similar to prior work [56, 57], we apply a Naive Bayesian classifier [26] on the description text and comment text factors to determine keywords that are associated with re-opened and non-reopened bugs. We use a Naive Bayesian classifier since it is commonly used to process text in spam filters due to its simplicity, its linear computation time and accuracy [58]. For this, we use a training set that is made up of two-thirds randomly selected bug reports. The Bayesian classifier is trained using two corpora that are derived from the training set. One corpus contains the description and comment text of the re-opened bugs[2] and the other corpus contains the description and comment text of the bugs that were not re-opened. The content of the description and comments are divided into tokens, where each token represents a single word. Since the bug comments often contain different types of text (e.g., code snippets), we did not stem the words or remove stop words. Prior work showed that stemming and removing stop words has very little influence on the final results [42].

The occurrence of each token is calculated and each token is assigned a probability of being attributed to a re-opened or none re-opened bug. These probabilities are based on the training corpus. Token probabilities are assigned based on how far their spam probability is from a neutral 0.5. If a token has never been seen before, it is assigned a probability of 0.4. The reason for assigning a low probability to new tokens is that they are considered innocent. The assumption here is that token associated with spam (or in our case re-opened bugs) will be familiar.

The probabilities of the highest 15 tokens are combined into one [27], which we use as a score value that indicates whether or not a bug will be re-opened. The choice of 15 tokens is motivated by prior work (e.g., [22]). Generally, the number of tokens provides a tradeoff between accuracy and complexity (i.e., having too little tokens may not capture enough information and having too many tokens may make the models too complex). A score value close to 1 indicates that the bug is likely not to be re-opened and vice versa. The score values of the description and comment text are then used in the decision tree instead of the raw text.

Dealing with imbalance in data: One issue that many real-world applications (e.g., in vision recognition [28], bioinformatics [29], credit card fraud detection [30] and bug prediction [31]) suffer from is data imbalance. What this means is that one class (i.e., majority) usually appears a lot more than another class (i.e., minority). This causes the decision tree to learn factors that affect the majority class without trying to learn about factors that affect the minority class. For example, in Eclipse the majority class is non-reopened bugs which has 1,284 bugs and the minority class is re-opened bugs, which contains 246 bugs. If the decision tree simply predicts that non of the bugs will be re-opened, then it will be correct 83.9% of the time (i.e., $\frac{1284}{1530}$). We discuss this observation in more detail in Section 6.

[2] For re-opened bugs, we used all the comments posted before the bugs were re-opened.

To deal with this issue of data imbalance, we must increase the weight of the minority class. A few different approaches have been proposed in the literature:

1. **Re-weighting the minority class:** Assigns a higher weight to each bug report of the minority class. For example, in our data, we would give a weight of 5.2 (i.e., $\frac{1284}{246}$) to each re-opened instance.
2. **Re-sampling the data:** Over- and under- sampling can be performed to alleviate the imbalance issue. Over-sampling increases the minority class instances to become at the same level as the majority class. Under-sampling decreases the majority class instances to reach the same level as the minority class. Estabrooks and Japkowicz [32] recommend performing both under- and over-sampling, since under-sampling may lead to useful data being discarded and over-sampling may lead to over-fitted models.

We built models using both re-weighting and re-sampling using the AdaBoost algorithm [33] available in the WEKA machine learning framework [34]. We performed both over- and under-sampling on the training data and predicted using a non-balanced test data set. We did the same using the re-weighting approach. Using re-sampling achieves better prediction results, therefore we decided to only use this in all our experiments. A similar finding was made in previous work [22].

It is important to note here that we re-sampled the training data set only. The test data set was not re-sampled or re-weighted in any way and maintained the same ratio of re-opened to non-re-opened bugs as in the original data set.

5 Case Study Results

In this section, we present the results of our case study on the Eclipse Platform 3.0, Apache HTTP server and OpenOffice projects. We aim to answer the two research questions posed earlier. To answer the first question we perform a Top Node analysis [12, 13] using each of the dimensions in isolation (to determine the best factors within each dimension) and using all of the dimensions combined (to determine the best factors across all dimensions). Then, we use these dimensions to build decision trees that accurately predict whether or not a bug will be re-opened.

Q1. Which of the extracted factors indicate, with high probability, that a bug will be re-opened?

We perform *Top Node analysis* to identify factors that are good indicators of whether or not a bug will be re-opened. In Top Node analysis, we examine the top factors in the decision trees created during our 10 x 10-fold cross validation. The most important factor is always the root node of the decision tree. As we move down the decision tree, the factors become less and less important. For example, in Figure 2, the most important factor in the tree is time_days. As we move down to level 1 of the decision tree, we can see that last_state and num_fix_files are the next important factors and so on. In addition to the level of the tree that a factor appears in, the occurrence of a factor at the different levels is also important. The higher the occurrence is, the stronger confidence is of the importance of that factor.

Team Dimension

Table 4 presents the results of the Top Node analysis for the team dimension. For the **Eclipse** project, the reporter name and the fixer name are the most important factors in the team dimension. Out of the 100 decision trees created (i.e., 10 x 10-fold cross validation), reporter name is the most important in 51 trees and fixer name was the most important in the remaining 49 trees. In **Apache** the reporter name is the most important factor in all 100 decision trees created for the team dimension. On the other hand, in **OpenOffice** the fixer name is the most important factor in all 100 decision trees.

Our finding shows that the reporter name and the fixer name are the most important factors in the team dimension. This indicates that some reporters and developers are more likely to have their bugs re-opened than others. The fixer experience is also important, ranking highly in level 1 of the decision trees of the Eclipse and OpenOffice projects.

It is important to note that in level 1 of the tree presented in Table 4, the frequencies of the attributes sum up to more than 200 (which would be the case when the attributes used were binary). This is because the Fixer name and reporter name variables are of type string and are converted to multiple nominal variables. Therefore, the frequencies of the attributes at level 1 of the tree sum up to more than 200.

This effect also made it hard to understand the concrete effect of the most important factors in each project. For example, a decision tree would say if developer A is the fixer and the developer experience is $>$ 10 bugs, then the bug is re-opened. Another branch of the tree might say if developer B is the fixer ad the developer experience is $>$ 10 bugs, then the bug is not re-opened. In such a case, it is difficult to determine the effect of the developer experience factor.

Therefore, in addition to examining the decision trees, we generated logistic regression models and used the coefficients of the factors to qualify the effect of the factor on a bug being re-opened [51]. In particular, we report the odds ratios of the analyzed factors. Odds ratios are the exponent of the logistic regression coefficients and indicate the increase to the likelihood of a bug being re-opened that 1 unit increase of the factor value causes. Odds ratios greater than 1 indicate a positive relationship between the independent (i.e., factors) and dependent variables (i.e., an increase in the factor value will cause an increase in the likelihood of a bug being re-opened). Odds ratios less than 1 indicate a negative relationship, or in other words, an increase in the independent variable will cause a decrease in the likelihood of the dependent variable.

Different fixer and reporter names were associated with different effects on a bug being re-opened. For the sake of privacy, we do not mention the effect of specific fixer and reporter names, and only discuss the effect of the fixer and reporter experience on a bug being re-opened.

In Eclipse, we found negative, but weak, effect between the reporter experience (odds ratio 0.99) and developer experience (odds ratio 0.99) and the likelihood of a bug being re-opened. This means that more experienced reporters and developers are less likely to have their bugs re-opened. In Apache we also found a negative and weak effect on reporter experience (odds ratio 0.99) and the likelihood of a bug being re-opened. In OpenOffice negative and weak effect between the developer experience (odds ratio 0.99) and the likelihood of a bug being re-opened.

Table 4 Top Node Analysis of the Team Dimension

	Eclipse		Apache		OpenOffice	
Level	**#**	**Attribute**	**#**	**Attribute**	**#**	**Attribute**
0	51	Reporter name	100	Reporter name	100	Fixer name
	49	Fixer name				
1	315	Fixer experience	1098	Fixer name	1490	Fixer experience
	277	Reporter name	1013	Fixer experience	480	Reporter name
	248	Reporter experience	862	Reporter experience	176	Reporter experience
	202	Fixer name				

Work Habit Dimension

Table 5 shows the results of the Top Node analysis for the work habit dimension. In **Eclipse** and **Apache**, the day of the year and the day of the month the bug was closed on were the most important factors. In the 100 decision trees created, the day of the year was the most important factor 76 and 97 times for Eclipse and Apache, respectively. For Apache, another important factors (i.e., in level 1) is the weekday that the bugs were closed in. For **OpenOffice**, the week day factor is the most important factor in 90 of the decision trees built for the OpenOffice project. The time factor was the most important in 10 of the 100 decision trees. Similar to the Eclipse and Apache projects, the day of the year was also important for the OpenOffice project (i.e, in level 1).

Examining the effect of the important factors for Eclipse showed that bugs closed later on in the year (i.e., day of the year) are more likely to be re-opened (odds ratio 1.05). We also find that bugs reported later in the month are less likely to be re-opened (odds ratio 0.95). In Apache, bugs closed later in the year have a negative, but weak, effect on bug re-opening (odds ratio 0.99). In OpenOffice, we found a negative relationship to bug re-opening for bugs closed on all days of the week (odds ratios in the range of 0.79 - 0.99), except for Wednesday where there was a positive relationship (odds ratio 1.03). For time, we find a weak and positive relationship between bugs closed in the morning (odds ratio 1.07) or at night (odds ratio 1.04) with bug re-opening.

Table 5 Top Node Analysis of the Work Habit Dimension

	Eclipse		Apache		OpenOffice	
Level	**#**	**Attribute**	**#**	**Attribute**	**#**	**Attribute**
0	76	Day of year	97	Day of year	90	Week day
	20	Month day	3	Month day	10	Time
	2	Time				
	1	Week day				
	1	Month day				
1	39	Day of year	73	Week day	412	Day of year
	31	Month day	68	Month day	134	Time
	22	Month	30	Day of year	73	Month
	20	Time	7	Time	28	Month day
	15	Week day	3	Month	23	Week day
	6	Day of year				

Bug Fix Dimension

Table 6 presents the Top Node analysis results for the bug fix dimension. For **Eclipse**, the time days factor, which counts the number of days it took from the time the bug was opened until its initial closure (i.e., the time it took to initially resolve the bug), is the most important factor in the fix dimension in 90 of the 100 decision trees. The number of files touched to fix the bug and the last status the bug was in when it was closed were the most important factor in 5 of the 100 trees. In the case of **Apache** and **OpenOffice**, the last status the bug was in when it was closed (i.e., before it was reopened) was the most important factor in all 100 decision trees. Also important are the time days and number of files in the fix, as shown by their importance in levels 1 of the decision trees.

As for the effect of the factors, in Eclipse we found that there exists a positive, but very weak, effect between the number of days it takes to close a bug (odds ratio 1.0) and the likelihood of a bug being re-opened. In addition, we found that bugs in the "resolved_duplicate", "resolved_worksforme" and "resolved_invalid" states before their final closure had the strongest chance of being re-opened. This means that when a bug is in any of those three aforementioned states before being closed, it should be closely verified since it is likely that it will be re-opened.

For Apache, bugs in the "resolved_duplicate", "resolved_wontfix", "resolved_invalid", "verified_invalid" and "resolved_worksforme" states were the most likely to be re-opened. In OpenOffice, we found that bugs in the "resolved_duplicate", "verified_wontfix", "verified_invalid" and "verified_worksforme" states prior to being closed were the most likely to be re-opened.

Table 6 Top Node Analysis of the Bug Fix Dimension

	Eclipse		**Apache**		**OpenOffice**	
Level	**#**	**Attribute**	**#**	**Attribute**	**#**	**Attribute**
0	90	Time days	100	Last status	100	Last status
	5	No. fix files				
	5	Last status				
1	93	No. fix files	261	Time days	293	Time days
	57	Last status	210	No. fix files	62	No. fix files
	38	Time days				

Bug Report Dimension

The Top Node analysis results of the bug report dimension are shown in Table 7. For the **Eclipse** project, the comment text content included in the bug report factor is the most important in this dimension, showing up as the most important in 94 of the 100 decision trees.

We examine the words that appear the most in the description and comments of the bugs. These are the words the Naive Bayesian classifier associates with re-opened and not re-opened bugs. Words such as "control", "background", "debugging", "breakpoint", "blocked" and "platforms" are associated with re-opened bugs. Words such as "verified", "duplicate", "screenshot", "important", "testing" and "warning" are associated with bugs that are not re-opened.

To shed some light on our findings, we manually examined 10 of the re-opened bugs. We found that three of these re-opened bugs involved threading issues. The discussions of these re-opened bugs talked about running processes in the "background" and having "blocked" threads. In addition, we found that bugs that involve the debug component were frequently re-opened, because they are difficult to fix. For example, we found comments such as "*Verified except for one part that seems to be missing: I think you forgot to add the...*" and "*This seems more difficult that[than] is[it] should be. I wonder if we can add...*".

For **Apache**, the description text is the most important factor in the bug report dimension. This finding is contrary to our observation in the Eclipse project, in which the comment text is shown to be the most important factor. However, the comment text is also of importance in the Apache project, appearing in level 1 of the decision trees.

The words that the Naive Bayesian classifier associates with re-opened bugs included words such as "cookie", "session", "block" and "hssfeventfactory" are associated with re-opened bugs. Words such as "attachment", "message", "ant", "cell" and "code" are associated with bugs that are not re-opened.

Manual examination of 10 of the re-opened bugs shows that four of the re-opened bugs in Apache are related to compatibility issues. For example, in one of the bugs examined, the bug was re-opened because the initial fix cloned a piece of code but did not modify the cloned code to handle all cases in its context. We found comments that said "*...The fix for this bug does not account for all error cases. I am attaching a document for which the code fails...*". A later comment said "*...I see what's missing. We borrowed the code from RecordFactory.CreateRecords but forgot to handle unknown records that happen to be continued...*".

In another example, the bug was being deleted because it was difficult to fix, and even after a fix was done, it did not fix the bug entirely. One of the comments that reflects the difficulty of the bug says *"...This bug is complex to fix, and for this reason will probably not be fixed in the 4.0.x branch, but more likely for 4.1. This will be mentioned in the release notes..."*. After the bug was initially fixed, a later comment attached by the developer who re-opened the bug says *"While the new session is now being created and provided to the included resource, no cookie is being added to the response to allow the session to be retrieved when direct requests to the included context are recieved..."*

Similar to the Eclipse project, in **OpenOffice** the comment text is the most important factor in the bug report dimension. The description text and the number of comments are also shown to be important, appearing in level 1 of the decision trees.

The Naive Bayesian classifier associates words such as "ordinal", "database", "dpcc", "hsqldb" and "sndfile" with re-opened bugs, whereas words such as "attached", "menu", "button", "wizard" and "toolbar" were associated with bugs that are not re-opened.

A manual examination of 10 of the re-opened bugs shows that seven of the re-opened bugs in OpenOffice are related to database access issues. In one of the examined bugs, the issue was related to a limitation of the HSQL database engine being used, where the reporter says *"Fields (names, data types) in HSQL-based tables cannot be modified after the table has been saved.."*. The bug is closed since this seemed to be a limitation of the HSQL database engine, as reported in this comment *"that would be nice, however I think it's (currently) a limitation of the used hsqldb database engine"*. Later on, support was added and the bug was re-opened and fixed.

Table 7 Top Node Analysis of the Bug Report Dimension

	Eclipse		**Apache**		**OpenOffice**	
Level	**#**	**Attribute**	**#**	**Attribute**	**#**	**Attribute**
0	94	Comment text	100	Description text	100	Comment text
	6	Description text				
1	181	Description text	105	Comment text	101	Description text
	16	Comment text	89	No. of comments	84	No. of comments
	1	Severity changed	1	Description size	9	No. in CC list
					5	Comment text
					1	Comment size

All Dimensions

Thus far, we looked at the dimensions in isolation and used Top Node analysis to determine the most important factors in each dimension. Now, we combine all of the dimensions together and perform the Top Node analysis using all of the factors. The Top Node analysis of all dimensions is shown in Table 8.

In **Eclipse**, the comment text is determined to be the most important factor amongst all of the factors considered in this study. In addition, the description text content is the next most important factor. For **Apache**, the last status of the bug when it is closed is the most important factor, followed by the description and comment text. Similar to the Eclipse project, in **OpenOffice** the comment text is shown to be the most important factor. The next most important factor is the last status factor, which appears in level 1 of the decision trees.

The comment text is the most important factor for the Eclipse and OpenOffice projects, while the last status is the most important one for Apache.

Table 8 Top Node Analysis Across All Dimensions

	Eclipse		**Apache**		**OpenOffice**	
Level	**#**	**Attribute**	**#**	**Attribute**	**#**	**Attribute**
0	96	Comment text	100	Last status	100	Comment text
	4	Description text				
1	180	Description text	280	Description text	200	Last status
	11	Comment text	132	Comment text		
	3	Severity changed	2	Time		
	1	Time	19	Month		
			1	Description size		
			10	Month day		
			8	No. of fix files		
			37	No. of comments		
			1	Severity		
			3	Fixer experience		

Q2. Can we accurately predict whether a bug will be re-opened using the extracted factors?

Following our study on which factors are good indicators of re-opened bugs, we use these factors to predict whether a bug will be re-opened. First, we build models that use only one dimension to predict whether or not a bug will be re-opened. Then, all of the dimensions are combined and used to predict whether or not a bug will be re-opened.

Table 9 shows the prediction results produced using decision trees. The results are the averages of the 10 times 10-fold cross validation. The variance of the measures is shown in brackets besides each average value. Ideally, we would like to obtain high precision, recall and F-measure values, especially for the re-opened bugs.

Eclipse: Out of the four dimensions considered, the bug report dimension was the best performing. It achieves a re-opened precision of 51.3%, a re-opened recall of 72.5% and 59.5% re-opened F-measure. The bug report dimension was also the best performer for not-reopened bugs; achieving a not re-opened precision of 94.3%, not re-opened recall of 86.2% and 89.9% not re-opened F-measure. The overall accuracy achieved by the bug report dimension is 83.9%. The rest of the dimensions did not perform nearly as well as the bug report dimension.

To put our prediction results for re-opened bugs in perspective, we compare the performance of our prediction models to that of a random predictor. Since the re-opened class is a minority class that only occurs 16.1% of the time, a random predictor will be accurate 16.1% of the time. Our prediction model achieves 51.3% precision, which is approximately a three-fold improvement over a random prediction. In addition, our model achieves a high recall of 72.5%.

Apache: The bug fix dimension is the best performing dimension. It achieves a re-opened precision of 40.1%, a re-opened recall of 89.8% and F-measure of 55.4%. The bug fix dimension also achieves the best not re-opened precision of 99.2%, recall of 90.7% and F-measure of 94.7%. The overall accuracy of the bug fix dimension is 90.6%.

Combining all of the dimensions improves the re-opened precision to 52.3%, the re-opened recall to 94.1% and the re-opened F-measure to 67.2%. Furthermore, combining all of the dimensions improves the not re-opened precision to 99.6%, recall to 94.1% and F-measure to 96.7%. The overall accuracy is improved to 94.0%.

In the case of Apache, re-opened bugs appear in only 6.5% of the total data set. This means that our re-opened precision improves over the random precision by more than 8 times. At the same time, we are able to achieve a very high recall of 94.1%.

OpenOffice: Similar to the Eclipse project, the bug report dimension is the best performing dimension for the OpenOffice project. The re-opened precision is 63.4%, the re-opened recall is 87.3% and the re-opened F-measure is 71.3%. The not re-opened precision is 93.0%, recall of 83.2% and not re-opened F-measure of 87.6%. The overall accuracy of the bug report dimension is 82.7%.

Using all of the dimensions in combination improves the re-opened precision to 78.6% (a three-fold improvement over a random predictor), the re-opened recall to 89.3% and the re-opened F-measure to 83.6%. The not re-opened precision improves to 95.9%, the not re-opened recall improves to 91.3% and the not re-opened F-measure improves to 93.6%. The overall accuracy improves to 90.8%.

Final remarks

As shown in Table 9, the precision varies across projects. For example, for Eclipse the precision is 52.1%, whereas for OpenOffice it is 78.6%. One factor that influences the precision value of prediction models is the ratio of re-opened to not re-opened bug reports [53]. This ratio is generally used a a baseline precision value [52,54]. Table 3 shows that the baseline precision for Eclipse is $\frac{246}{1530} = 16.1\%$, whereas the baseline precision for OpenOffice is $\frac{10572}{40173} = 26.3\%$. Therefore, we expect our prediction models to perform better in the case

Table 9 Prediction results

	Dimension	Re-opened Precision	Re-opened Recall	Re-opened F-measure	Not Re-reopened Precision	Not Re-opened Recall	Not Re-opened F-measure	Accuracy
Eclipse	Team	18.3(0.13) %	45.5(1.1) %	25.9(0.26) %	85.4(0.05) %	61.0(0.29) %	71.0(0.15) %	58.5(0.19) %
	Work habit	21.8(0.14) %	54.1(1.2) %	30.9(0.28) %	87.7(0.07) %	62.5(0.36) %	72.8(0.19) %	61.2(0.23) %
	Fix	18.9(0.06) %	68.2(2.0) %	29.5(0.17) %	88.2(0.13) %	44.1(0.94) %	58.1(0.70) %	47.9(0.46) %
	Bug	**51.3(0.74) %**	**72.5(0.81) %**	**59.5(0.48) %**	**94.3(0.03) %**	**86.2(0.22) %**	**89.9(0.07) %**	**83.9(0.15) %**
	All	52.1(0.99) %	70.5(0.69) %	59.4(0.55) %	93.9(0.02) %	86.8(0.21) %	90.2(0.07) %	84.2(0.15) %
Apache	Team	8.4(0.01) %	46.7(0.29) %	14.4(0.02) %	94.7(0.00) %	65.3(0.04) %	77.2(0.02) %	64.1(0.03) %
	Work habit	7.0(0.01) %	38.3(0.32) %	11.9(0.03) %	93.9(0.00) %	65.1(0.02) %	76.9(0.01) %	63.4(0.02) %
	Fix	**40.1(0.08) %**	**89.8(0.12) %**	**55.4(0.08) %**	**99.2(0.00) %**	**90.7(0.01) %**	**94.7(0.00) %**	**90.6(0.01) %**
	Bug	28.5(0.06) %	73.3(0.18) %	40.9(0.08) %	97.9(0.00) %	87.2(0.02) %	92.2(0.01) %	86.3(0.02) %
	All	52.3(0.09) %	94.1(0.05) %	67.2(0.07) %	99.6(0.00) %	94.1(0.01) %	96.7(0.00) %	94.0(0.00) %
OpenOffice	Team	57.5(0.02) %	71.9(0.03) %	63.8(0.01) %	88.8(0.00) %	81.0(0.01) %	84.8(0.00) %	78.6(0.01) %
	Work habit	50.2(0.01) %	75.6(0.03) %	60.3(0.01) %	89.4(0.00) %	73.2(0.02) %	80.5(0.01) %	73.9(0.01) %
	Fix	44.0(0.01) %	**87.3(0.04) %**	58.5(0.01) %	**93.0(0.01) %**	60.3(0.04) %	73.1(0.02) %	67.4(0.01) %
	Bug	**63.4(0.01) %**	81.4(0.02) %	**71.3(0.01) %**	92.6(0.00) %	**83.2(0.01) %**	**87.6(0.00) %**	**82.7(0.00) %**
	All	78.6(0.02) %	89.3(0.01) %	83.6(0.01) %	95.9(0.00) %	91.3(0.01) %	93.6(0.00) %	90.8(0.00) %

of OpenOffice compared to Eclipse. Another factor that influences the variation in prediction results is the fact that we are using the same factors for all projects. In certain cases, some factors may perform better for some projects than others.

Overall, our results show that fairly accurate prediction models can be created using a combination of the four dimensions. However, although combining all of the dimensions provides a considerable improvement over using the best single dimension for the Apache and OpenOffice projects, it only provides a slight improvement for the Eclipse project. Having a predictor that can perform well without the need to collect and calculate many complex factors makes it more attractive for practitioners to adopt the prediction approach practice.

We can build explainable prediction models that can achieve a precision of 52.1-78.6% and a recall of 70.5-94.1% recall when predicting whether a bug will be re-opened and a precision between 93.9-99.6% and recall of 86.8-94.1% when predicting if a bug will not be re-opened.

6 Comparison with Other Prediction Algorithms

So far, we used decision trees to predict whether a bug will be re-opened. However, decision trees are not the only algorithm that can be used. Naive Bayes classifier and Logistic regression are two very popular algorithms that have been used in many prediction studies (e.g., [9]). In this section, we compare the prediction results of various prediction algorithms that can be used to predict whether or not a bug will be re-opened. In addition, we used the prediction from the Zero-R algorithm as a baseline for the prediction accuracy. The Zero-R algorithm simply predicts the majority class, which is not re-opened in our case.

The prediction results using the different algorithms are shown in Table 10. Since different algorithms may provide a tradeoff between precision and recall, we use the F-measure

Table 10 Results using different prediction algorithms

	Algorithm	**Re-opened Precision**	**Re-opened Recall**	**Re-opened F-measure**	**Not Re-reopened Precision**	**Not Re-opened Recall**	**Not Re-opened F-measure**	**Accuracy**
Eclipse	Zero-R	NA	0 %	0 %	83.9(0.00) %	100(0.00) %	91.3(0.00) %	83.9(0.00) %
	Naive Bayes	49.0(0.33) %	**73.9(0.81) %**	58.7(0.34) %	**94.5(0.03) %**	85.1(0.10) %	89.5(0.03) %	83.3(0.08) %
	Logistic Reg	45.3(0.41) %	67.2(0.77) %	53.8(0.35) %	93.1(0.03) %	84.1(0.15) %	88.1(0.15) %	81.4(0.10) %
	C4.5	**52.1(0.99) %**	70.5(0.69) %	**59.4(0.55) %**	93.9(0.02) %	**86.8(0.21) %**	**90.2(0.07) %**	**84.2(0.15) %**
Apache	Zero-R	NA	0 %	0 %	93.5(0.00) %	100(0.00) %	96.7(0.00) %	93.5(0.00) %
	Naive Bayes	46.5(0.10) %	69.8(0.35) %	55.7(0.10) %	97.8(0.01) %	**94.4(0.01) %**	96.1(0.00) %	92.8(0.00) %
	Logistic Reg	46.5(0.15) %	78.3(0.23) %	58.2(0.14) %	98.4(0.01) %	93.7(0.01) %	96.0(0.00) %	92.7(0.01) %
	C4.5	**52.3(0.09) %**	**94.1(0.05) %**	**67.2(0.07) %**	**99.6(0.00) %**	94.1(0.01) %	**96.7(0.00) %**	**94.0(0.00) %**
OpenOffice	Zero-R	NA	0 %	0 %	73.7(0.00) %	100(0.00) %	84.8(0.00) %	73.7(0.00) %
	Naive Bayes	48.7(0.06) %	**90.5(0.02) %**	63.3(0.04) %	95.1(0.00) %	65.7(0.13) %	77.7(0.06) %	72.3(0.06) %
	Logistic Reg	69.8(0.01) %	88.9(0.01) %	78.2(0.01) %	95.6(0.00) %	86.3(0.01) %	90.7(0.00) %	86.9(0.00) %
	C4.5	**78.6(0.02) %**	89.3(0.01) %	**83.6(0.01) %**	**95.9(0.00) %**	**91.3(0.01) %**	**93.6(0.00) %**	**90.8(0.00) %**

to compare the different prediction algorithms. As expected, the Zero-R algorithm achieves the worst performance, since it does not detect any of the re-opened bugs. The Naive Bayes algorithm performs better in some cases. For example, for the Eclipse project the Naive Bayes algorithm achieves a re-opened recall of 73.9% and not re-opened precision of 94.5%. The Logistic Regression model performs slightly worse achieving re-opened F-measure of 53.8% (precision: 45.3%, recall: 67.2%) and not re-opened F-measure of 88.1% (precision: 93.1%, recall: 84.1%). Decision trees perform slightly worse (in some cases) than Naive Bayes for Eclipse, Apache and OpenOffice. More importantly however is that decision trees provide explainable models. Practitioners often prefer explainable models since it helps them understand why the predictions are the way they are.

7 Discussion: Commit vs. Bug Work Habits

Our work habits factors are based on the time that the bug was initially closed. The reason for using the time the bug was initially closed was due to the fact that we wanted to investigate whether developers were more inclined to close bugs during specific times (e.g., to reduce their work queue). However, another side that may contribute to bug re-opening is the work habits factors of the commit or change performed to the address the bug. For example, commits made specific times may be associated with a higher change of a bug being re-opened later on.

To examine the effect of the commit work habits dimension, we extract the same factors for the work habits dimension for each commit. The factors are shown in Table 11. Since not all bugs could be linked to a commit and we need all of the factors to perform our analysis, our dataset reduced in size. For Eclipse, we were able to link 1,144 bugs where 187 were re-opened and 957 were not. For OpenOffice, we were able to link 19,393 bugs where 7181 were re-opened and 12,212 were not. For Apache were were only able to link 278 bug reports. After examination of the linked data, we decided to perform the analysis for Eclipse and OpenOffice, but not for Apache (i.e., due to the low number of linked bug reports).

First, we redo the top node analysis of the work habits factors, this time including both bug closing and commit work habits. The results are shown in Table 12. To be clear, we

Table 11 Commit Work Habits Factors

Dim	Factor	Type	Explanation	Rationale
Change work habits	Change time	Nominal	Time in hours (Morning, Afternoon, Evening, Night) when the change to address the bug was made.	Changes made to address bugs at certain times in the day (e.g., late afternoons) are more/less likely to be re-opened.
	Change weekday	Nominal	Day of the week (e.g., Mon or Tue) when the change to address the bug was first made.	Changes made to address bugs on specific days of the week (e.g., Fridays) are more/less likely to be re-opened.
	Change month day	Numeric	Calendar day of the month (0-30) when the changes to address the bug was made.	Changes made to address bugs at specific periods like the beginning, mid or end of the month are more/less likely to be re-opened.
	Change month	Numeric	Month of the year (0-11) when the change to address the bug was made.	Changes made to address bugs in specific months (e.g., during holiday months like December) are more/less likely to be re-opened.
	Change day of year	Numeric	Day of the year (1-365) when the change to address the bug was made.	Changes made to address bugs in specific times of the year(e.g., later on in the year) are more/less likely to be re-opened.

explicitly label each factor with its association to bugs or commits, shown in brackets. We observe that for both projects, the work habits factors from the initial bug closure are the most important factors. In particular, the day of year and the month were the most important. Commit work habits factors are placed in level 1 of the decision tress, showing that they are also important, but clearly less important than the initial bug closure.

To examine whether including the commit work habit factors improves prediction accuracy, we present the prediction results in Table 13. For each project, the first row presents the prediction results when the bug work habits are only considered. The second row presents the prediction results when the commit and bug work habits factors are combined. We see that for both, Eclipse and OpenOffice, the prediction results improve. For Eclipse, we see an improvement of 3.8% in re-opened precision and 4.5% in re-opened recall, whereas, for OpenOffice we see an improvement in prediction results of 19.6% in re-opened precision and 15.4% improvement in re-opened recall.

8 Threats to Validity

In this section, we discuss the possible threats to validity of our study.

Threats to Construct Validity consider the relationship between theory and observation, in case the measured variables do not measure the actual factors. Some of the re-opened bugs considered in our study were re-opened more than once. In such cases, we predict for the first time the bug was re-opened. In future studies, we plan to investigate bugs that are re-opened several times.

One of the attributes used in the People dimension is the fixer name. We extracted the names of the fixers from the committed CVS changes. In certain cases, the fixer and the committer of the changes are two different people. In the future, we plan to use heuristics that may improve the accuracy of the fixer name factor.

For the work habits dimension, we use the dates in the Bugzilla bug tracking system. These times refer to the server time and may not be the same as the local user time. Furthermore, we do not take timezone information into account since such information is not available. These threats show that in some cases, our work habits may not directly measure the commit habits of developers.

Table 12 Top Node Analysis of the Commit and Bug Work Habit Dimension

	Eclipse		OpenOffice	
Level	**#**	**Attribute**	**#**	**Attribute**
0	66	(Bug) Day of year	100	(Bug) Day of year
	21	(Bug) Month		
	5	(Bug) Time		
	3	(Commit) Month		
	2	(Bug) Week day		
	2	(Commit) Time		
	1	(Commit) Month day		
1	16	(Bug) Week day	106	(Commit) Day of year
	5	(Commit) Month day	94	(Commit) Month
	11	(Bug) Month		
	25	(Commit) Month		
	36	(Bug) Day of year		
	9	(Commit) Weekday		
	12	(Bug) Time		
	20	(Bug) Month day		
	37	(Commit) Day of year		
	14	(Commit) Time		

Table 13 Prediction Results when Considering Commit Work Habits

	Dimension	Re-opened Precision	Re-opened Recall	Re-opened F-measure	Not Re-reopened Precision	Not Re-opened Recall	Not Re-opened F-measure	Accuracy
Eclipse	Work habit (bug only)	20.7(0.17) %	50.6(1.2) %	29.2(0.32) %	86.5(0.07) %	61.7(0.47) %	71.8(0.26) %	59.9(0.32) %
	Work habit (bug and commit)	24.5(0.23) %	55.1(1.5) %	33.8(0.43) %	88.4(0.08) %	66.7(0.33) %	75.9(0.17) %	64.7(0.24) %
OpenOffice	Work habit (bug only)	62.7(0.04) %	75.4(0.06) %	68.4(0.02) %	83.6(0.01) %	73.5(0.07) %	78.2(0.02) %	74.2(0.02) %
	Work habit (bug and commit)	82.3(0.02) %	87.3(0.02) %	84.7(0.01) %	92.3(0.00) %	88.9(0.01) %	90.6(0.00) %	88.3(0.01) %

Threats to Internal validity refers to whether the experimental conditions makes a difference or not, and whether there is sufficient evidence to support the claim being made.

The percentage of bug reports that met the prerequisites to be included in our study is small (e.g., for Eclipse we were able to extract a total of 18,312 bug reports, of which 1,530 met our prerequisites). At first glance, this seems to be a low bug reports used to extracted bug reports ratio. However, such a relatively low ratio is a common phenomenon in studies using bug reports [9, 10]. In addition, we would like to note that the percentage of open-to-reopened bugs in the data set used and the original data set are quite close. For example, in the Eclipse project, 16.1% of the bug reports were re-opened, whereas 10.2% of the bug reports were re-opened in the original data set.

We use 24 different factors that cover four dimensions to predict re-opened bugs. Although this set of factors is large, it is only a subset of factors that may be used to predict re-opened bugs. Other factors such as social networks factors for example can be used to further improve the prediction results.

Bird *et al.* [48] showed that bias due to imperfect linking between historical databases is common and may impact the performance of prediction techniques.

Threats to External Validity consider the generalization of our findings. In this study, we used three large, well established Open Source projects to conduct our case study. Although these are large open source projects, our results may not generalize (and as we have seen do not generalize) to all open source or commercial software projects.

We use decision trees to perform our prediction and compared our results to 3 other popular prediction algorithms. Decision trees performed well, for all three projects, compared to the 3 algorithms we compared with, however, using other prediction algorithms may produce different results. One major advantage to using decision trees is that they provide explainable models that practitioners can use to understand the prediction results.

In our manual examination of the re-opened bugs, we only examined a very small sample of 10 bug reports. The purpose of this analysis was to shed some light on the type of information that we were able to get from the re-opened bug reports. Our findings do not generalize to all re-opened bugs.

9 Related Work

We divide the related work into two parts: the work related to the dimension used and work related to bug report quality and triage.

9.1 Work Related to Dimensions Used

The work closest to this paper is the work by Zimmermann *et al.* [55] which characterizes and predicts re-opened bugs in Windows. The authors perform a qualitative and quantitative study and find that some of the reasons for bug reopens are the fact that bugs were difficult to reproduce, developers misunderstood the root cause, bug reports had insufficient information and the fact that priority of the bug may have been initially underestimated. Through their quantitative analysis, the authors find that bugs reported by customers or found during system testing are more likely to be re-opened. Also, bugs that are initially assigned to someone on a different team or geographic location are more likely to be re-opened. In many ways this paper complements our study since we both focus on bug reopens. However, our study is done on OSS projects, whereas Zimmermann *et al.* use commercial systems. Also, their study surveys Microsoft developers and provides more insight about the reasons for bug re-opens at Microsoft.

Work habit dimension: Anbalagan and Vouk [17] performed a case study on the Ubuntu Linux distribution and showed that the day of the week on which a bug was reported impacts the amount of time it will take to fix the bug. Śliwerski et al. [16] measured the frequency of bug introducing changes on different days of the week. Through a case study on the Eclipse and Mozilla projects, they showed that most bug introducing changes occur on Fridays. Hassan and Zhang [12] used the time of the day, the day of the week and the month day to predict the certification results of a software build and Ibrahim *et al.* [22] used the time of the day, the week day and the month day that a message was posted to predict whether a developer will contribute to that message. Eyolfson *et al.* [50] examine the effect of time of day and developer experience on commit bugginess in two open source projects. The authors find that approximately 25% of commits are buggy, that commits checked in between 00:00 and 4:00 AM are more likely to be buggy, developers who commit on a daily basis write less-buggy commits and bugginess for commits per day of the week vary for different projects. No prediction was performed.

The work habit dimension extracts similar information to those used in the aforementioned related work. However, our work is different in that we use the information to investigate whether these work habit factors affect the chance of a bug being re-opened.

Bug report dimension: Mockus *et al.* [18] and Herraiz *et al.* [35] used information contained in bug reports to predict the time it takes to resolve bugs. For example, in [18], the authors showed that in the Apache and Mozilla projects, 50% of the bugs with priority P1 and P3 were resolved within 30 days and half of the P2 bugs were resolved within 80 days. On the other hand, 50% of the bugs with priority P4 and P5 took much longer to resolve (i.e., their resolution time was in excess of 100 days). They also showed that bugs logged against certain components were resolved faster than others.

Similar to the previous work, we use the information included in bug reports, however, we do not use this information to study the resolution time of a bug. Rather, we use this information to predict whether or not a bug will be re-opened.

Bug fix dimension: Hooimeijer *et at.* [19] built a model that measures bug report quality and predicts whether a developer would choose to fix the bug report. They used the total number of attachments that are associated with bug reports as one of the features in the model. Similarly, Bettenburg *et al.* [36] used attachment information to build a tool that recommends to reporters how to improve their bug report. Hewett and Kijsanayothin [37] used the status of a bug (e.g., Worksforme) as one of the features to model the bug resolution time.

Similar to the previous studies, we use information about the initial bug fix as input into our model, which predicts whether or not a bug will be re-opened.

People dimension: Schröter *et al.* [38] analyzed the relationship between human factors and software reliability. Using the Eclipse bug dataset, they examined whether specific developers were more likely to introduce bugs than others. They observed a substantial difference in bug densities in source code developed by different developers. Anvik *et al.* [39] and Jeong *et al.* [40] were interested in determining which developers were most suitable to resolve a bug.

We use the names and the experience of the bug reporters and fixers to predict whether or not a bug will be re-opened. Although our paper is similar to other previous work in terms of the factors used, to the best of our knowledge, this paper is the first to empirically analyze whether or not a bug will be re-opened.

9.2 Work on Bug Report Quality and Triage

Antoniol *et al.* [49] use decision trees, naive bayes and logistic regression to correctly classify issues in bug tracking systems as bugs or enhancements. Bettenburg *et al.* [36] investigate what makes a good bug report. They find that there is a mismatch between information that developers need (i.e., stack traces, steps to reproduce and test cases) and what users supply. Aranda and Venolia [44] report a field study of coordination activities related to bug fixing. They find that data stored in repositories can be incomplete since they do not take into account social, organizational and technical knowledge. Bettenburg *et al.* [45] examine the usefulness of duplicate bug reports and find that duplicate bug reports contain valuable information that can be combined with other bug reports. Guo *et al.* [46] chaterize factors that affect whether a bug is fixed in Windows Vista and Windows 7. They find that bugs reported by people with better reputation, and on the same team or within the same geographic proximity are more likely to get fixed.

Another line of work aims to assist in the bug triaging process. This work focused on predicting who should be assigned to fix a particular bug [39], the analysis of bug report reassignments [40] and predicting the severity of bug reports [47]. In contrast to prior work, in this work we focus on re-opened bugs.

This paper is an extension of an earlier conference version of the paper [41] in which we conduct our study on two additional Open Source systems, Apache and OpenOffice. We also contrast the findings from the three projects, in terms of which factors best predict re-opened bugs and their prediction accuracy, and report our findings. We provide insight about the way in which the most important factors impact the likelihood of a bug being re-opened and examine work habits of the commit.

10 Conclusion

Re-opened bugs increase maintenance costs, degrade the overall user-perceived quality of the software and lead to unnecessary rework by busy practitioners. Therefore, practitioners are interested in identifying factors that influence the likelihood of a bug being re-opened to better deal with, and minimize the occurrence of re-opened bugs. In this paper, we used information extracted from the bug and source code repositories of the Eclipse, Apache and OpenOffice open source projects to derive 24 different factors, which make up four different dimensions, to predict whether or not a bug will be re-opened. We performed Top Node analysis to determine which factors are the best indicators of a bug being re-opened. The Top Node analysis showed that the factors that best indicate re-opened bugs depends on the project. The comment text is the most important factor for the Eclipse and OpenOffice projects, while the last status is the most important one for Apache. In addition, we provide insight about the way in which the important factors impact the likelihood of a bug being re-opened. Then, with the derived factors, we can build explainable prediction models that can achieve 52.1-78.6% precision and 70.5-94.1% recall when predicting whether a bug will be re-opened. The findings of this work contributes towards better understanding of what factors impact bug re-openings so they can be carefully examine. Doing so will reduce the number of re-opened bugs and the maintenance costs associated with them.

Acknowledgments

This research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B), 22700033, 2010 by the Ministry of Education, Culture, Sports, Science and Technology, Japan. In addition, it is supported in part by research grants from the Natural Science and Engineering Research Council of Canada.

References

1. L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, no. 3, pp. 17–23, 2000.
2. T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, 2007, p. 9.

3. M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 99, no. 6, pp. 864–878, 2009.
4. M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," *Reverse Engineering, Working Conference on*, vol. 0, pp. 135–144, 2009.
5. T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions of Software Engineering*, vol. 26, no. 7, pp. 653–661, July 2000.
6. R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*, 2008, pp. 181–190.
7. C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista," *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pp. 518–528, 2009.
8. N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*, 2008, pp. 521–530.
9. L. D. Panjer, "Predicting eclipse bug lifetimes," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007, p. 29.
10. C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007, p. 1.
11. S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, 2006, pp. 173–174.
12. A. E. Hassan and K. Zhang, "Using decision trees to predict the certification result of a build," in *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, 2006, pp. 189–198.
13. J. Sayyad and C. Lethbridge, "Supporting software maintenance by mining software update records," in *ICSM '01: Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, 2001, p. 22.
14. "Bugzilla," http://www.bugzilla.org/.
15. "Bugzilla a bug's life cycle," https://bugs.eclipse.org/bugs.
16. J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" in *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, 2005, pp. 1–5.
17. P. Anbalagan and M. Vouk, ""days of the week" effect in predicting the time taken to fix defects," in *DEFECTS '09: Proceedings of the 2nd International Workshop on Defects in Large Software Systems*, 2009, pp. 29–30.
18. A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, no. 3, pp. 309–346, 2002.
19. P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 34–43.
20. A. E. Hassan, "Predicting faults using the complexity of code changes," in *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 78–88.
21. J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.
22. W. M. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. E. Hassan, "Should i contribute to this discussion?" in *MSR '10: Proceedings of the 2010 international working conference on Mining software repositories*, 2010.
23. S. Kim, J. E. James Whitehead, and Y. Zhang, "Classifying software changes: Clean or buggy?" *IEEE Transactions on Software Engineering*, vol. 34, pp. 181–196, 2008.
24. B. Efron, "Estimating the error rate of a prediction rule: Improvement on cross-validation," *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.
25. W. Shang, Z. M. Jiang, B. Adams, and A. E. Hassan, "Mapreduce as a general framework to support research in mining software repositories (MSR)," in *MSR '09: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2009, p. 10.
26. T. A. Meyer and B. Whateley, "SpamBayes: Effective open-source, Bayesian based, email classification system," *Proceedings of the First Conference on Email and Anti-Spam*, 2004.
27. P. Graham, "A plan for spam," 2002.
28. J. S. Sánchez, R. Barandela, A. I. Marqués, and R. Alejo, "Performance evaluation of prototype selection algorithms for nearest neighbor classification," in *SIBGRAPI '01: Proceedings of the XIV Brazilian Symposium on Computer Graphics and Image Processing*, 2001, p. 44.
29. R. Barandela, J. S. Sánchez, V. García, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition*, vol. 36, no. 3, pp. 849–851, 2003.

30. P. Chan and S. J. Stolfo, “Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection,” in *In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 164–168.
31. J. Sayyad and C. Lethbridge, “Supporting software maintenance by mining software update records,” in *ICSM ’01: Proceedings of the IEEE International Conference on Software Maintenance (ICSM’01)*, 2001, p. 22.
32. A. Estabrooks and N. Japkowicz, “A mixture-of-experts framework for learning from imbalanced data sets,” in *IDA ’01: Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, 2001, pp. 34–43.
33. Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” 1995.
34. I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition).* Morgan Kaufmann Publishers Inc., 2005.
35. I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, “Towards a simplification of the bug report form in eclipse,” in *MSR ’08: Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 145–148.
36. N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “What makes a good bug report?” in *SIGSOFT ’08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.
37. R. Hewett and P. Kijsanayothin, “On modeling software defect repair time,” *Empirical Softw. Engg.*, vol. 14, no. 2, pp. 165–186, 2009.
38. A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, “If your bug database could talk...” in *ISESE ’06: Proceedings of the 5th International Symposium on Empirical Software Engineering. Volume II: Short Papers and Posters*, 2006, pp. 18–20.
39. J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?” in *ICSE’06: Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.
40. G. Jeong, S. Kim, and T. Zimmermann, “Improving bug triage with bug tossing graphs,” in *ESEC/FSE ’09: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 111–120.
41. E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, K. Matsumoto, “Predicting Re-opened Bugs: A Case Study on the Eclipse Project” in *WCRE’10: Proceedings of the 17th Working Conference on Reverse Engineering*, 2010, pp. 249–258.
42. I. Androutsopoulos, J. Koutsias and K. V. Cb and C. D. Spyropoulos, “An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages” in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, 2000, pp. 160–167.
43. M. Fischer, M. Pinzger and H. Gall, “Populating a Release History Database from Version Control and Bug Tracking Systems” in *ICSM’03: Proceedings of the International Conference on Software Maintenance*, 2003, pp. 23–32.
44. J. Aranda and G. Venolia, “The secret life of bugs: Going past the errors and omissions in software repositories” in *ICSE ’09: Proceedings of the 31st International Conference on Software Engineering*, 2009, pp. 298–308.
45. N. Bettenburg, R. Premraj, T. Zimmermann and S. Kim, “Duplicate bug reports considered harmful really?” in *ICSM ’08: Proceedings of International Conference on Software Maintenance*, 2008, pp. 337-345.
46. P. J. Guo and T. Zimmermann N. Nagappan and B. Murphy, “Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows” in *ICSE ’10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, 2010, pp. 495-504.
47. A. Lamkanfi, S. Demeyer, E. Giger and B. Goethals, “Predicting the severity of a reported bug” in *ICSE ’06: Proceedings of the IEEE Working Conference on Mining Software Repositories*, 2010, pp. 1- 10.
48. C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov and P. Devanbu, “Fair and Balanced? Bias in Bug-Fix Datasets” in *ESEC/FSE’09: Proceedings of the the Seventh joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2009, pp. 121- 130.
49. G. Antoniol, K. Ayari, M. Di Penta, F. Khomh and Y. Guéhéneuc, “Is it a bug or an enhancement?: a text-based approach to classify change requests” in *CASCON ’08: Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008, pp. 304-318.
50. J. Eyolfson, L. Tan and P. Lam, “Do Time of Day and Developer Experience Affect Commit Bugginess?” in *MSR ’11: Proceedings of the 8th Working Conference on Mining Software Repositories*, 2008, pp. 153-162.

51. A. Mockus, “Organizational volatility and its effects on software defects” in *FSE ’10: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2010, pp. 117-126.
52. T. Lee, J. Nam and D. Han, S. Kim and H. In, “Micro interaction metrics for defect prediction” in *ESEC/FSE ’11: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011, pp. 311-321.
53. T. Menzies, A. Dekhtyar, J. Distefano and J. Greenwald, “Problems with Precision: A Response to Comments on Data Mining Static Code Attributes to Learn Defect Predictors” *IEEE Transactions on Software Engineering*, vol. 33, pp. 637–640, 2007.
54. E. Shihab, A. Mockus, Y. Kamei, B. Adams and A.E. Hassan, “High-impact defects: a study of breakage and surprise defects” in *ESEC/FSE ’11: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011, pp. 300-310.
55. T. Zimmermann, N. Nagappan, P. J. Guo and B. Murphy, “Characterizing and Predicting Which Bugs Get Reopened” in *ICSE ’12: Proceedings of the 34th International Conference on Software Engineering*, 2012, pp. 495-504.
56. O. Mizuno and H. Hata, “An integrated approach to detect fault-prone modules using complexity and text feature metrics” in *Proceedings of the 2010 international conference on Advances in computer science and information technology*, 2010, pp. 457-568.
57. O. Mizuno, S. Ikami, S. Nakaichi and T. Kikuno, “Spam Filter Based Approach for Finding Fault-Prone Software Modules” in *MSR’07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007, pp. 4-8.
58. E. Michelakis, I. Androutsopoulos, G. Paliouras, G. Sakkis and P. Stamatopoulos, “Filtron: A Learning-Based Anti-Spam Filter” in *PROCEEDINGS OF THE 1ST CONFERENCE ON EMAIL AND ANTI-SPAM*, 2004.