Predicting Re-opened Bugs: A Case Study on the Eclipse Project

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

*Software Analysis and Intelligence Lab (SAIL)
Queen's University
Kingston, ON, Canada
Email: {emads, kamei, walid, bram, ahmed}@cs.queensu.ca

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

***Abstract*—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.**

In this paper, we study and predict re-opened bugs through a case study on the Eclipse project. We structure our study along 4 dimensions: 1) the work habits dimension (e.g., the weekday on which the bug was initially closed on), 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). Our case study on the Eclipse Platform 3.0 project shows that the comment and description text, the time it took to fix the bug, and the component the bug was found in are the most important factors in determining whether 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 62.9% precision and 84.5% recall when predicting whether a bug will be re-opened.

I. 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 was dedicated to addressing issues related to software bugs. For example, software defect prediction work used 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 focused on predicting the time it takes to fix a bug [9]–[11].

Previous work treated all bugs equally, meaning, they 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 time to resolve. 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. In addition, such re-opened bugs may lead end users to lose trust in the quality of the software product, which negatively impacts the overall end user's experience. To the best of our knowledge, this is the first work to study re-opened bugs. We perform a case study on the Eclipse Platform 3.0 project to determine factors that indicate whether a bug will be re-opened. Knowing which factors are attributed to re-opened bugs prepares practitioners 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 (in terms of fixing and testing) to such bugs and their fixes.

We combine data extracted from the bug and source control repositories to extract 22 factors that are grouped into four different dimensions: 1) work habits dimension, 2) bug report dimension 3) bug fix dimension and 4) team dimension. 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 factors indicate, with high probability, that a bug will be re-opened?** The bug description and comment text, the number of days it took to make the initial bug fix and the component the bug was found in are the best indicators of whether or not a bug will be re-opened.

1095-1350/10 $26.00 © 2010 IEEE
DOI 10.1109/WCRE.2010.36

Q2 Can we accurately predict, using the four dimensions, if a bug will be re-opened using the extracted factors? We use 22 different factors to build highly accurate prediction models that predict whether or not a bug will be re-opened. Our model can correctly predict whether a bug will be re-opened with 62.9% precision and 84.5% recall.

The rest of the paper is organized as follows. Section II describes the life cycle of a bug. Section III presents the methodology of our study. We detail our data processing steps in Section IV. The case study results are presented in Section V. We compare the prediction results using different algorithms in Section VI. The threats to validity and related work are presented in Sections VII and VIII, respectively. Section IX concludes the paper.

II. 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 reported bugs, such as the time the bug was reported and the component the bug was found in. The information stored in bug tracking systems is leveraged by many researchers to investigate different phenomena.

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 closed.

A diagram representing the majority of the states bugs go through is shown in Figure 1. When developers, testers or user's 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*).

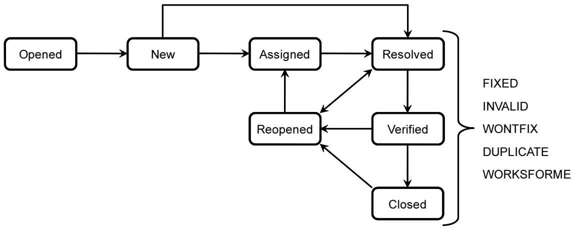

Fig. 1. Bug resolution process [15]

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.

III. Approach

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 we employ to evaluate the performance of the prediction models.

A. 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.

The factors we extracted cover four different dimensions. We describe each dimension and its factors in more detail.

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 I. The table explains the rationale behind choosing each attribute.

[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 I
FACTORS CONSIDERED IN OUR STUDY

Dimension	Factor	Type	Data Source	Explanation	Rationale
Work habits	Time	Numeric	Bug database	Time in hours (0-23) when the bug was closed. For re-opened bugs, we used the time of the first bug closure.	Bugs closed at certain times in the day (e.g., late afternoons) are more/less likely to be re-opened
	Weekday	Nominal	Bug database	Day of the week (e.g., Mon, Tue, etc.) when the bug was closed. For re-opened bugs, we used the time of the first bug closure.	Bugs closed on specific days of the week (e.g., Fridays) are more/less likely to be re-opened
	Month day	Numeric	Bug database	Calendar day of the month (0-30) when the bug was closed. For re-opened bugs, we used the time of the first bug closure.	Bugs closed at specific periods (i.e., beginning, mid or end of the month) in the month are more/less likely to be re-opened
	Month	Numeric	Bug database	Month of the year (0-11) when the bug was closed. For re-opened bugs, we used the time of the first bug closure.	Bugs closed in specific months (e.g., during holiday months like December) are more/less likely to be re-opened
Bug report	Component	Nominal	Bug database	Component the bug was found in (e.g., UI, Debug, Search)	Certain components might be harder to fix; bugs found in these components are more/less likely to be re-opened
	Platform	Nominal	Bug database	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	Bug database	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. For re-opened bugs we use the severity before the first re-open	Bugs with high severity values are harder to fix and are more likely to be re-opened
	Priority	Numeric	Bug database	Priority of the reported bug. A low priority value (i.e., 1) indicates an important bug and a high priority (i.e., 5) indicates a bug of low importance. For re-opened bugs we use the priority before the first re-open	Bugs with low priority value (i.e., high importance) are likely to get careful attention and have a smaller chance of being re-opened
	Number in CC list	Numeric	Bug database	Number of persons in the cc list of the logged bugs. For re-opened bugs, we only count the number of persons in the cc list 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	Bug database	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	Bug database	The text content of the bug description	Words included in the bug description can indicate whether the bug is more likely to be re-opened
	Number of comments	Numeric	Bug database	The number of comments attached to a bug report. For re-opened bugs, we only include the number of comments 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	Bug database	The number of words in all the comments attached to the bug report. For re-opened bugs, we only include the comments 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	Bug database	The text content of all the comments attached to the bug report. For re-opened bugs, we only include the comments 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	Bug database	States whether the priority of the bug was changed after the initial report. For re-opened bugs, we only consider the priority change if it occurred before the first re-open	Bugs that have their priorities increased are generally followed more closely and are less likely to be re-opened
Bug fix	Time days	Numeric	Bug database	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. Hence the time it takes to fix a bug is a good indicator if it will be re-opened.
	Last status	Nominal	Bug database	The last status of the bug when it is closed. For re-opened bugs, we record the status of the first closure	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	Bug and source code databases	The number of files edited to fix the bug. For re-opened bugs, we only consider the initial fix	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	Bug database	Name of the bug reporter	Bugs reported by specific individuals are more/less likely to be re-opened
	Fixer Name	String	Bug and source code databases	Name of the bug fixer. For re-opened bugs, we use the name of the person who performed the initial fix	Bugs fixed by specific individuals are more/less likely to be re-opened
	Reporter Experience	Numeric	Bug database	The number of bugs reported by the bug reporter before reporting this bug	More experienced reporters are less likely to have their bugs re-opened
	Fixer Experience	Numeric	Bug and source code databases	The number of bug fixes the fixer performed before fixing this bug	Bugs fixed by experienced fixers are less likely to be re-opened

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 I.

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 I 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. In some cases, multiple changes are grouped into one and committed as one big change. Analysis of our data showed no evidence of this issue. The majority of the changes used in our study are 1 or 2 files in size and the largest change had 92 files.

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 I, is made up of four factors that cover bug reporters, bug fixers and their experience.

The four dimensions and their factors listed in Table I 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.

B. 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.

TABLE II
CONFUSION MATRIX

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

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 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. Similar to previous studies [22], in our case study, we set this minimum node size to be 10 to mitigate noise in our predictions.

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.

C. 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 II shows an example of a confusion matrix.

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.

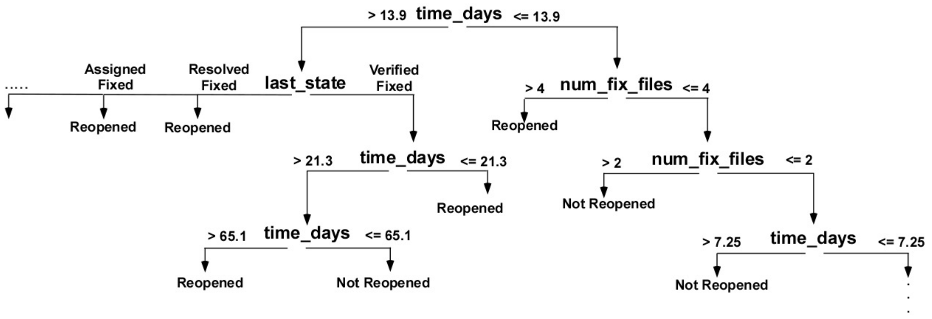

Fig. 2. Sample decision tree

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:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \quad (1)$$

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 insight, 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 as F-measure(nre) $= \frac{2*P(nre)*R(nre)}{P(nre)+R(nre)}$ for bugs that are not re-opened.

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 divided into two parts: testing data that contains one-tenth of the original data set and training data that contains the rest of the data set. The model is trained using the training data and its accuracy is tested using the testing data. We repeat the 10-fold cross validation 10 times and report the average of the 10 runs.

IV. Data processing

The Eclipse Platform 3.0 project was used as a case study since it is a large and mature Open Source Software (OSS) project with a large user base and a rich development history. We leveraged two data sources from the Eclipse Platform 3.0 project, the bug database and the source code control (CVS) logs.

To extract data from the Eclipse bug database, we wrote a script that crawls and extracts bug report information from Eclipse's online Bugzilla database. 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 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.

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. 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 bugs reports studied, 246 were re-opened and 1,284 were not.

For each bug report, we extract 22 different factors that cover four different dimensions, described in Table I. Most of the factors were directly derived from the bug or code databases. However, two factors in the bug report dimension are text based and required special processing. 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. For this, we use a training set that is made up of 2/3 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. The occurrence of each token is calculated and each token is assigned a probability of being attributed to a re-opened or non re-opened bug. The probabilities of the highest 15 tokens are combined into one [27], which we use as a score value that indicates whether a bug will be re-opened or not. A score value close to 1 indicates that the bug is likely not to be re-opened and vice-versa. The score values are then used in the decision tree instead of the raw description and comment 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 our case 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 all bugs will not be re-opened, then it will be correct 83.9% of the time (i.e., $\frac{1284}{1530}$).

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 become at 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.

V. Case Study Results

In this section, we present the results of our case study on the Eclipse Platform 3.0 project. 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 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 a bug will be re-opened or not. In Top Node analysis, we examine the top factors in the decision trees created during our 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.

The Top Node analysis for the **team dimension** is shown in Table III. The Fixer name and Reporter name are the most important factors in the Team dimension. Out of the 10 decision trees created, Fixer name was the most important factor in 5 of them and Reporter name in the other 5. It is important to note that in level 1 of the tree presented in Table III, the frequencies of the attributes sum up to more than 20 (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 20.

As for the **work habit dimension**, the Month and Time (i.e., 0-23 hr) of the day the bug was closed on were the most important factors, as depicted in Table IV. In the 10 decision trees created, the month was the most important factor 9 times and the time factor was the most important factor once.

Table V presents the Top Node analysis results for the **bug fix dimension**. 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

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

the most important factor in the fix dimension. In all of the 10 decision trees we built, the Time days factor was the most important in 7 of them and the Last status the bug held before it was reopened was the most important factor in the remaining 3 trees.

The Top Node analysis results of the **bug report dimension** are shown in Table VI. The comment text content included in the bug report factor is the most important in this dimension. Out of the 10 decision trees we built, the comment text content was the most important in all of them.

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.

We manually examined some of the re-opened bugs. We found that many of the 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...*".

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 VII. 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, the time to perform the fix in days and the component the bug is logged against are also important factors in determining whether a bug will be re-opened or not.

The comment text, description text, time it took to fix the bug and the component the bug was logged against are the most important factors in determining whether or not a bug will be re-opened.

Q2. Can we accurately predict, using the four dimensions, if 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 III
TOP NODE ANALYSIS OF THE TEAM DIMENSION

Level	Frequency	Attribute
0	5	Fixer name
	5	Reporter name
1	35	Fixer experience
	28	Reporter name
	24	Reporter experience
	19	Fixer name
2	50	Fixer experience
	40	Reporter experience
	1	Fixer name

TABLE IV
TOP NODE ANALYSIS OF THE WORK HABIT DIMENSION

Level	Frequency	Attribute
0	9	Month
	1	Time
1	7	Week day
	6	Time
	4	Month
	2	Month day
2	13	Week day
	11	Month day
	8	Month
	2	Time

Table VIII shows the prediction results produced using decision trees. The results are the averages of 10 runs. Ideally, we would like to obtain high precision, recall and F-measure values, especially for the re-opened bugs. Out of the four dimensions considered, the bug report dimension was the best performing. It achieves a re-opened precision of 62.9%, a re-opened recall of 82.5% and 71.2% re-opened F-measure. The bug report dimension was also the best performer for not-reopened bugs; achieving a not re-opened precision of 96.5%, not re-opened recall of 90.7% and 93.5% not re-opened F-measure. The overall accuracy achieved by the bug report dimension is 89.3%. The rest of the dimensions did not perform nearly as well as the bug report dimension.

To provide a full context of the results, we provide the confusion matrix values (average of 10 runs) when all dimensions are used in combination. The TP= 21, the FN= 4, the FP= 13 and the TN= 115.

Using all of the dimensions in combination slightly reduced the overall accuracy, but it improved the re-opened recall. Overall, we do not see a significant improvement using all of the dimensions compared to using the bug report dimension alone. This finding shows that fairly accurate prediction models can be created from the bug report information that is readily available in most bug tracking systems. 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 such methods in practice.

TABLE V
TOP NODE ANALYSIS OF THE FIX DIMENSION

Level	Frequency	Attribute
0	7	Time days
	3	Last status
1	8	No. fix files
	6	Last status
	6	Time days
2	12	No. fix files
	12	Time days

TABLE VI
TOP NODE ANALYSIS OF THE BUG DIMENSION

Level	Frequency	Attribute
0	10	Comment text
1	19	Description text
	1	Component
2	9	Comment text
	4	Description size
	3	No. of comments
	3	No. in CC List
	2	Comments size
	2	Description text
	2	Priority changed
	1	Platform

We can build explainable prediction models that can achieve 62.9% precision and 84.5% recall when predicting whether a bug will be re-opened and 96.8% precision and 89.6% recall when predicting if a bug will not be re-opened.

VI. COMPARISON WITH OTHER PREDICTION ALGORITHMS

In this paper, 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 IX. 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, achieving a re-opened F-measure of 59.9% (precision: 50.5%, recall: 74.4%) and not re-opened F-measure of 89.9% (precision: 94.5%, recall: 85.6%). The logistic regression model performs slightly better achieving re-opened F-measure of 64.4% (precision: 55.1%, recall: 78.5%) and not re-opened F-measure of 91.3% (precision: 95.5%, recall: 87.5%). Decision trees outperformed all of the other prediction methods, shown in bold. 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.

TABLE VII
TOP NODE ANALYSIS ACROSS ALL DIMENSIONS

Level	Frequency	Attribute
0	10	Comment text
1	16	Description text
	2	Component
	1	Time days
2	8	Time days
	5	No. of comments
	3	Reporter name
	2	Time
	1	Description size
	1	Description text
	1	Fixer name
	1	Month

VII. THREATS TO VALIDITY

In this section, we discuss the possible threats to validity of our study. In this study, we used a large, well established Open Source project to conduct our case study. Although the Eclipse Platform 3.0 project is a large open source project, our results may not generalize to other 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 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.

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.

We were able to extract a total of 18,312 bug reports, of which 1,530 met the prerequisites to be included in our study. 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 (16.1% in the data set used vs. 10.2% in the original data set).

VIII. RELATED WORK

We divide the related work into the four dimensions used in our study: the work habit dimension, the bug report dimension, the bug fix dimension and the people dimension.

TABLE VIII
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
Team	27.3 %	67.1 %	38.7 %	91.2 %	65.6 %	76.3 %	65.8 %
Work habit	33.0 %	74.0 %	45.5 %	93.4 %	70.5 %	80.2 %	71.0 %
Fix	20.9 %	**82.8 %**	33.2 %	92.8 %	39.2 %	54.1 %	46.3 %
Bug	**62.9** %	82.5 %	**71.2 %**	**96.5 %**	**90.7 %**	**93.5 %**	**89.3 %**
All	62.9 %	84.5 %	71.4 %	96.8 %	89.6 %	93.0 %	88.8 %

TABLE IX
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
Zero-R	NA	0 %	0 %	83.9 %	100 %	91.3 %	83.9 %
Naive Bayes	50.5 %	74.4 %	59.9 %	94.5 %	85.6 %	89.9 %	83.6 %
Logistic Reg	55.1 %	78.5 %	64.4 %	95.5 %	87.5 %	91.3 %	86.0 %
C4.5	**62.9 %**	**84.5 %**	**71.4 %**	**96.8 %**	**89.6 %**	**93.0 %**	**88.8 %**

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.

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 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 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.

IX. 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. In this paper, we used information extracted from the bug and source code repositories of a project to derive 22 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 comment text, description text, time to resolve the bug and the component the bug was found in are the top factors. Using the derived factors, we can build explainable prediction models that can achieve 62.9% precision and 84.5% recall when predicting whether a bug will be re-opened. In the future, we plan to extend this study to include more projects and examine whether the factors that influence a bug being re-opened in Eclipse, affect other projects in a similar way.

Acknowledgments

This research is being conducted as a part of the Next Generation IT Program and Grant-in-aid for Young Scientists (B), 20700028, 2009 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, 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.