2014 IEEE 38th Annual International Computers, Software and Applications Conference

Automated Configuration Bug Report Prediction Using Text Mining

Xin Xia*‡, David Lo†, Weiwei Qiu*, Xingen Wang*, and Bo Zhou*§
*College of Computer Science and Technology, Zhejiang University
†School of Information Systems, Singapore Management University
xxkidd@zju.edu.cn, davidlo@smu.edu.sg, {qiuweiwei, newroot, bzhou}@zju.edu.cn

***Abstract*—Configuration bugs are one of the dominant causes of software failures. Previous studies show that a configuration bug could cause huge financial losses in a software system. The importance of configuration bugs has attracted various research studies, e.g., to detect, diagnose, and fix configuration bugs. Given a bug report, an approach that can identify whether the bug is a configuration bug could help developers reduce debugging effort. We refer to this problem as *configuration bug reports prediction*. To address this problem, we develop a new automated framework that applies text mining technologies on the natural-language description of bug reports to train a statistical model on historical bug reports with known labels (i.e., configuration or non-configuration), and the statistical model is then used to predict a label for a new bug report. Developers could apply our model to automatically predict labels of bug reports to improve their productivity. Our tool first applies feature selection techniques (e.g., information gain and Chi-square) to preprocess the textual information in bug reports, and then applies various text mining techniques (e.g., naive Bayes, SVM, naive Bayes multinomial) to build statistical models. We evaluate our solution on 5 bug report datasets including accumulo, activemq, camel, flume, and wicket. We show that naive Bayes multinomial with information gain achieves the best performance. On average across the 5 projects, its accuracy, configuration F-measure and non-configuration F-measure are 0.811, 0.450, and 0.880, respectively. We also compare our solution with the method proposed by Arshad et al.. The results show that our proposed approach that uses naive Bayes multinomial with information gain on average improves accuracy, configuration F-measure and non-configuration F-measure scores of Arshad et al.'s method by 8.34%, 103.7%, and 4.24%, respectively.**

***Keywords*—Configuration Bug, Data Mining, Feature Selection**

I. Introduction

Modern software systems allow users to customize the systems behaviors via configuration options. However, the flexibility of configuration options could potentially affect the reliability of the software systems. Previous studies show that configuration bugs (i.e., misconfiguration) are one of the major causes for the downtime of large-scale software systems [1]. For example, Barroso and Hölzle reported that configuration bugs contributed to approximately 28% of service-level failures, which is the second major cause of failures at one of Google's main data centers [2]. Facebook reported that a configuration bug blocked their 500 million users to access its website for several hours [3]. Similar findings are also reported for other software systems, such as Microsoft Azure [4], Amazon EC2 [5], and Google [6]. Hale reported that more than 80% of network outages are caused due to configuration bugs [7].

Aside from causing downtime, configuration bugs also increase cost in various ways. Kappor concluded that technical support consumed 17% of the total cost of maintaining working desktop computers in a company, and troubleshooting configuration errors (i.e., bugs) took a large proportion of it [8]. Yin et al. showed that 27% of bugs are configuration-related in a major storage company's customer support database [9].

Recently, much research effort has been made to detect, diagnose, and fix configuration bugs [10], [11], [12]. Wang et al. propose PeerPressure which identifies configuration errors by leveraging statistical analysis [10]. Zhang and Ernst develop a tool named ConfDiagnoser which combines static analysis, dynamic profiling, and statistical analysis to identify the root cause of a configuration bug [11]. Xu et al. propose SPEX to automatically infer configuration constraints from source code, and then use these constraints to expose misconfiguration vulnerabilities, and detect error-prone configuration design [12].

To further advance the state-of-the-art in this area, and help developers to reduce debugging effort and improve their productivity, in this paper, we investigate a new research problem: given a bug report, identify whether this bug is a configuration bug. We refer to this problem as *configuration bug reports prediction*. We develop a new automated tool which applies text mining technologies on the natural-language description of bug reports to train a statistical model on the historical bug reports with known labels (i.e., configuration or non-configuration) to classify a new bug report as either a configuration bug report or a non-configuration bug report. The rationale of our tool is to explore the valuable natural-language information provided in bug reports. Although bug reporters do not manually flag a bug as configuration-related, the natural-language description of the bug report could be enough to identify that the bug is indeed configuration-related.

The number of terms in natural-language description of bug reports are often large and this could introduce inaccuracy to the prediction of bug report labels (i.e., configuration or non-configuration). Thus, in our tool, we first apply feature selection techniques [13] to select significant terms from the bug reports. Two state-of-the-art feature selection techniques, information gain and Chi-square [13], are investigated in this paper. Next, we represent these bug reports by using the selected terms, and leverage classification techniques to build a

‡The work was done while the author was visiting Singapore Management University.
§Corresponding author.

0730-3157/14 $31.00 © 2014 IEEE
DOI 10.1109/COMPSAC.2014.17

statistical model (i.e., a classifier). In this paper, we investigate various state-of-the-art classification techniques (e.g., naive Bayes, SVM, naive Bayes multinomial, and kNN [14]) which are widely used in text mining and software engineering literatures, c.f., [14], [15], [16], [17], [18]. Finally, the statistical model is used to classify a new bug report as either a configuration or non-configuration bug report.

The most related work to ours is proposed by Arshad et al. [19]. They propose a keyword query-based approach to automatically identify configuration bugs. In their query, existence of keywords such as "config", "setting", "setup", "set-up" and "set up" are used to identify if a bug report is a configuration bug. We use their method as a baseline that we would compare our proposed approach with.

We evaluate our tool on 5 datasets from different open source software projects: accumulo[1], activemq[2], camel[3], flume[4], and wicket[5]. In total, we analyze 3,203 bug reports. We measure the performance of the approaches in terms of accuracy, configuration F-measure, and non-configuration F-measures. On average across the 5 datasets, naive Bayes multinomial with information gain achieves the best performance; its accuracy, configuration F-measure and non-configuration F-measure are 0.811, 0.450, and 0.880, respectively. The results show that naive Bayes multinomial with information gain on average improves accuracy, configuration F-measure and non-configuration F-measure scores of Arshad et al.'s method by 8.34%, 103.7%, and 4.24%, respectively.

The main contributions of this paper are:

1) We develop a new automated tool which applies text mining technologies on the natural-language description of bug reports to identify configuration bugs.
2) We experiment on a broad range of datasets containing a total of 3,203 bugs to demonstrate the effectiveness of our tool. We show that our tool outperforms the method proposed by Arshad et al. on the configuration bug report prediction problem by a substantial margin.

The remainder of the paper is organized as follows. We describe a motivating example in Section II. We outline the overall framework of our proposed approach in Section III. We elaborate the feature selection and classification techniques in Sections IV and V, respectively. We report our experiment results in Section VI. We describe related work in Section VII. We conclude and mention future work in Section VIII.

II. Motivation

A typical bug report contains various useful fields, such as status, priority, component, summary, and description. Notice that in the summary and description fields, reporters would use natural language to describe the bug. In this paper, we refer to the natural-language description of bug reports as the textual information in the summary and description fields. Figure 1 shows a bug report from accumulo with BugID=1560.[6]

[1]https://issues.apache.org/jira/browse/ACCUMULO
[2]https://issues.apache.org/jira/browse/AMQ
[3]https://issues.apache.org/jira/browse/CAMEL
[4]https://issues.apache.org/jira/browse/FLUME
[5]https://issues.apache.org/jira/browse/WICKET
[6]https://issues.apache.org/jira/browse/ACCUMULO-1560

Figure 2 shows the corresponding fix patch for this bug. The root cause of this bug is an incompatibility between different architectures and platforms, i.e., the RPM packages which were created for 64-bit amd64 architecture, are not recognized on Redhat Linux with x86_64 architecture. The fix of this bug is to modify its maven configuration file, i.e., pom.xml, by changing the value of the "needarch" parameter from "true" to "x86_64".

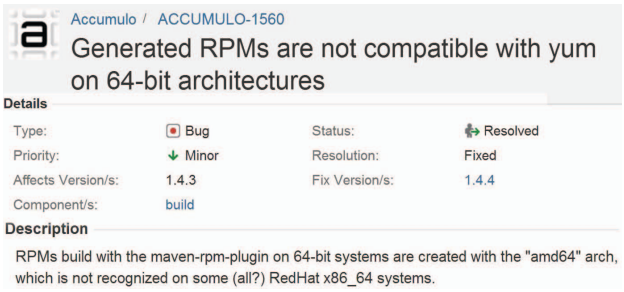

Fig. 1. Bug Report of accumulo with BugID=1560.

```
--- accumulo/branches/1.4/pom.xml       2013/07/10 00:44:08     1501623
+++ accumulo/branches/1.4/pom.xml       2013/07/10 00:59:40     1501624
@@ -201,7 +201,7 @@
          </description>
          <copyright>2011 The Apache Software Foundation.</copyright>
          <url>http://accumulo.apache.org</url>
-         <needarch>true</needarch>
+         <needarch>x86_64</needarch>
          <group>Utilities</group>
          <requires>
            <require>jdk</require>
```

Fig. 2. The Patch File for the Bug in Figure 1.

Observations and Implications. From the above motivating example, we can observe the following:

1) The natural-language description of a bug report provides information to indicate whether a bug is a configuration bug. For example, the natural-language description of the bug report in Figure 1 describes a RPM package compatibility problem due to a configuration bug when maven is used to build the system.
2) Some terms in a bug report are good indicators to identify whether it is a configuration bug, while some other terms are noise. For example, the terms "compatible", "build", "maven", "RPM", etc., in the bug report in Figure 1 are good indicators; while the terms "system", "plugin ", "create" are noise since both configuration and non-configuration bug reports would use these terms. Thus, it would be good to select good indicators (i.e., useful terms) and remove noise (i.e., useless terms) from a bug report.

The above observations tell us that we could use the natural-language description of bug reports to identify configuration bugs, and selecting good terms (indicators) from bug reports could help to improve classification performance. Therefore, an automated tool which applies text mining techniques on the natural-language description of bug reports could assist developers to identify configuration bugs and

reduce their debugging effort and cost. The following scenarios illustrate the benefits of our tool:

Scenario 1 - Without Tool: Bob is a junior developer in a large software project which contains many configuration files. One day, he was asked to fix a newly reported bug. Due to his lack of experience, he tried to fix this bug by modifying various source code files. And he wasted much time and effort to locate the relevant source code files, however, he still did not find the root cause of the bug. Then, he asked a senior developer Alice for help. After reading the description of the bug report, Alice told him that this bug was caused by a wrong setting in a configuration file, and only one parameter in the configuration file needs to be modified. Finally, Bob fixed this bug by making that small change in the configuration file, however much time and effort had been wasted.

Scenario 2 - With Tool: Bob is a junior developer in a large software project which contains many configuration files. One day, he was asked to fix a newly reported bug. Bob initially thought that the root cause of this bug was in one of the source code files. To confirm his initial analysis, he used our tool. However, our tool told Bob that this bug is highly likely to be a configuration bug, and thus he should look into the configuration files. He followed the direction of our tool and soon he was able to locate the root cause which is a wrong setting in a parameter in a configuration file. The total bug fixing process only took him a short period of time.

III. Overall Framework

Figure 3 shows our configuration bug report prediction framework. The whole framework includes two phases: model building phase and prediction phase. In the model building phase, our goal is to build a classifier (i.e., statistical model) by leveraging text mining techniques from historical bug reports with known labels (i.e., configuration or not). In the prediction phase, this classifier would be used to predict if an unknown bug report would be a configuration bug report or not (i.e., a non-configuration bug).

Our framework first extracts features from a set of training bug reports (i.e., bug reports with known status) (Step 1). Features are various quantifiable characteristics of bug reports that could potentially distinguish reports that are related to configuration bugs from those that are not. In this paper, we use textual features from the natural-language description of bug reports. Our framework extracts the description and summary texts from bug reports. For each description and summary text, our framework tokenizes them, removes stop words (e.g., I, you, he, the), stems them (i.e., reduces them to their root forms, e.g., "configuration" and "configure" are reduced to "config"), and represents them in the form of a "bag of words" [20].

Then, our framework applies feature selection techniques to select a subset of relevant textual features to further improve the prediction performance (Step 2).[7] By employing feature selection techniques, we can reduce model building times, and avoid overfitting [13]. In this paper, we investigate 2 state-of-the-art feature selection techniques, i.e., information gain and Chi-square.

[7]Detailed information of the feature selection techniques is presented in Section IV.

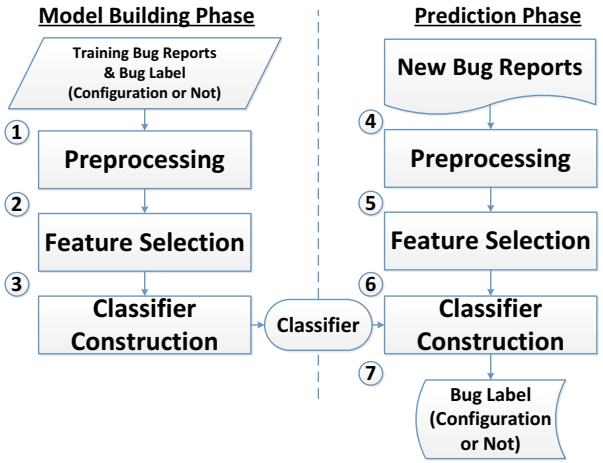

Fig. 3. Proposed Configuration Bug Report Prediction Framework.

After we select a subset of textual features, our framework next constructs a classifier (i.e., statistical model) based on the selected textual features of the training bug reports (Step 3).[8] A classifier is a statistical model which assigns labels (in our case: configuration or non-configuration) to a data point (in our case: a bug report) based on its textual features. The classifier construction phase would compare and contrast the features of bug reports that are configuration bugs, and those of bug reports that are not. In this paper, we investigate 5 text mining techniques, i.e., naive Bayes multinomial, naive Bayes, kNN, Bayesian network, and SVM [14].

In the prediction phase, the classifier is then used to predict whether a bug report with unknown label is a configuration bug or not. For each of such bug reports, our framework first preprocesses and extracts textual features from it (Step 4), and represent it by using the features selected in the model building phase (Step 5). Next, these features are input into the classifier in the classifier application step (Step 5). This step would output the prediction result which is one of the following labels: configuration or non-configuration (Step 6).

IV. Feature Selection Techniques

Previous studies show that feature selection techniques could improve the performance of text categorization [13], [21]. In this section, we describe 2 state-of-the-art feature selection techniques, i.e., information gain (IG) and Chi-square (CHI). Let us denote a bug report collection as $BR = \{(B_1, C_1)\ , (B_2, C_2), ...(B_N, C_N)\}$, where B_i represents the i^{th} bug report and C_i is a label that represents whether this bug report is a configuration bug (c) or not ($\bar{c}$) (i.e., $C_i \in \{c, \bar{c}\}$), and the terms in BR_i are denoted as $BR_i = \langle t_1, t_2, ..., t_{|BR_i|}\rangle$. For a term t, and the configuration label c, for a bug report B_i, there would be 4 possible relationships:

1) (t, c): B_i contains the term t, and it is a configuration bug (i.e., c);
2) $(t, \bar{c})$: B_i contains the term t, but it is not a configuration bug (i.e., $\bar{c}$);

[8]Detailed information of this step is presented in Section V.

3) $(\bar{t}, c)$: B_i does not contain the term t, but it is a configuration bug (i.e., c);
4) $(\bar{t}, \bar{c})$: B_i does not contain the term t, and it is not a configuration bug (i.e., $\bar{c}$);

Based on the above 4 possible relationships, we can compute information gain (IG) and chi-square (CHI) scores which are elaborated in the following paragraphs.

A. Information Gain (IG)

Information gain (IG) [13], [21] measures the number of bits of information required for predicting a label (i.e., configuration or non-configuration) by knowing the presence or absence of a term in a bug report. The information gain (IG) score of term t and label c is computed as:

$$IG(t,c) = \sum_{c' \in \{c,\bar{c}\}} \sum_{t' \in \{t,\bar{t}\}} p(t',c') \times log \frac{p(t',c')}{p(t') \times p(c')} \quad (1)$$

B. Chi-square (CHI)

Chi-square [13], [21] measures divergence from the chi-square distribution expected with one degree of freedom to judge extremeness if one assumes the occurrence of a term t is actually independent of the label c. We denote A as the number of bug reports where relationship (t, c) is observed, B as the number of bug reports where $(t, \bar{c})$ is observed, C as the number of bug reports where $(\bar{t}, c)$ is observed, and D as the number of bug reports where $(\bar{t}, \bar{c})$ is observed. Then, the chi-square (CHI) score of term t and label c is computed as:

$$CHI(t,c) = \frac{N \times (AD - CB)^2}{(A+C) \times (B+D) \times (A+B) \times (C+D)} \quad (2)$$

In the above equation, N is the total number of bug reports.

C. Ranking the Scores

After we apply feature selection techniques (e.g., IG or CHI) to compute the scores for each term, we rank these scores from high to low to generate a ranked list. The higher the score is, the more important the term to distinguish a label c is. We select top $k\%$ terms whose feature selection scores are in the top $k\%$ of the ranked list, and remove the other terms. In this way, we reduce the number of features in the model building phase, and also in the prediction phase. By default, we choose the top 10% of the total number of terms.

V. Classification Techniques

In this section, we elaborate 5 state-of-the-art classification techniques which we use in this paper.

A. Naive Bayes (NB)

Naive Bayes [22] assumes that features (i.e., terms) are conditionally independent given a label (configuration or non-configuration bug). Based on this assumption, for a bug report $BR_i = \langle t_1, t_2, ..., t_{|BR_i|} \rangle$, where t_i is the terms in the bug report, and a label C_i, we have:

$$p(BR_i|C_i) = \prod_{i=1}^{|BR_i|} p(t_i|C_i) \quad (3)$$

By applying Bayes Theorem on Equation (3), we have:

$$p(C_i = c|BR_i) = \frac{p(C_i = c) \times p(BR_i|C_i = c)}{\sum_{c' \in \{c,\bar{c}\}} p(C_i = c') \times p(BR_i|C_i = c')} = \frac{p(C_i = c) \times \prod_{i=1}^{|BR_i|} p(t_i|C_i = c)}{\sum_{c' \in \{c,\bar{c}\}} p(C_i = c') \times \prod_{i=1}^{|BR_i|} p(t_i|C_i = c')} \quad (4)$$

We can use Equation (4) to predict the label for a bug report BR_i, i.e., if $p(C_i = c|BR_i) \geq p(C_i = \bar{c}|BR_i)$, then we classify this bug report as a configuration bug report; else otherwise. The major advantage of naive Bayes classification is its short computational training time, since it assumes conditional independence. Notice that in naive Bayes, we only consider the presence or absence of a term in a bug report; the number of times a term appears in the bug report is not considered.

B. Naive Bayes Multinomial (NBM)

Naive Bayes multinomial (NBM) [22] is similar to naive Bayes, but the label for a bug report is not simply determined by the presence or absence of terms in the bug report, rather by the number of times each of the terms appears in the bug report. In general, NBM performs better than naive Bayes when the total number of unique terms in the bug report collection is large.

C. K-Nearest Neighbor (kNN)

K-nearest neighbor (kNN) [14] is an instance-based algorithm for text mining. The general idea behind kNN is to predict the label of a bug report based on its k-nearest neighbors (i.e., bug reports). kNN has two steps:

1) For an unlabeled bug report, kNN finds its k-nearest neighbors in the training bug reports according to a distance metric;
2) kNN then assigns to this unlabeled bug report the most frequent label that its k-nearest neighbors have, i.e., if the number of neighboring bug reports that are configuration bug reports are more than those that are not, then we classify it as a configuration bug report; else otherwise.

There are various distance metrics, such as Euclidean distance, Minkowsky distance, Manhattan distance, etc – c.f., [23]. In this work, by default, we use Euclidean distance as the distance metric.

D. Support Vector Machine (SVM)

Support vector machine (SVM) [14] was developed from statistical learning theory, and it constructs a hyperplane or a set of hyperplanes in a high- or infinite-dimensional space, which are used for classification. Each training bug report is represented as a point in a multi-dimensional space where each term (i.e., feature) represents a dimension. SVM selects a small number of critical boundary instances (i.e., bug reports) as support vectors for each label (in our case, the labels are configuration and non-configuration), and builds a linear or non-linear discriminant function to form decision boundaries with the principle of maximizing the margins among training bug reports belonging to the different labels.

E. Bayesian Network (BN)

Bayesian network (BN) is a graphical model which uses probability theory to represent the relationships between terms and labels in bug reports [24]. It is a directed acyclic graph (DAG) and each node in a BN represents either a term or the label. A directed edge between two nodes denotes that there is a causal (i.e., dependency) relationship between them. In the model building phase, BN would construct a Bayesian network from the training bug reports. In the prediction phase, this Bayesian network is then used to predict the label for a new unlabeled bug report.

VI. Experiments and Results

In this section, we evaluate the effectiveness of our proposed tool. The experimental environment is an Intel(R) Core(TM) i5 3.20 GHz CPU, 4GB RAM desktop running Windows 7 (32-bit). We first present our experiment setup, evaluation metrics, and 4 research questions in Section VI-A, VI-B, and VI-C, respectively. We then present our experiment results that answer the 4 research questions (Sections VI-D, VI-E, VI-F, and VI-G).

A. Experiment Setup

We evaluate our proposed tool on 6 datasets from different open source software projects: accumulo, activemq, camel, flume, trafficserver, and wicket. For each of the projects, we first download all the issue reports from their corresponding issue tracking systems (i.e., JIRA systems). Notice in JIRA, some issue reports are feature requests, and we remove them and only keep the issue reports whose types are "bug". We refer to these issue reports as bug reports. Next, we mine the projects' corresponding source code repositories (i.e., Git), and we find commits which are linked to our collected bug reports, i.e., these commits fixed bugs described in bug reports. To identify these commits, we automatically analyze their logs using a regular expression. In total, we collected 3,203 bug reports in these 6 projects which can be linked to commits that fix them. A feature of a configuration bugs is that its fix should include a modification to a configuration file. Thus, for each of the commits which is linked to a bug report, we also check the files that are modified by the commit. For a bug report, if its linked commits include the modification of a configuration file, we consider it as a configuration bug. We also manually analyze these bug reports and their corresponding fixes, to further confirm whether each of the bug reports is a configuration bug or not. In total, we have 524 configuration bug reports. Table I presents the statistics of the 6 projects. The columns correspond to the project name (Project), the number of bug reports collected (# Bugs), the time period of the collected bug reports (Time), the number of configuration bug reports (# Confs), and the number of unique terms (# Terms).

TABLE I. Statistics of Collected Datasets.

Project	# Bugs	Time	# Confs	# Terms
accumulo	181	2011. 10 - 2013. 06	33	227
activemq	175	2005. 12 - 2007. 12	29	327
camel	1,189	2007. 07 - 2013. 09	333	1,261
flume	279	2010. 07 - 2013. 05	83	341
wicket	1,379	2006. 11 - 2013. 09	46	1,340

We use WVTool [25] to extract terms from these bug reports. WVTool is a Java library for statistical language modeling, which is used to create word vector representations of text documents. We use WVTool to tokenize natural-language description of bug report, remove stop words, and do stemming. We remove terms which appear less than 5 times to reduce noise.

Stratified ten-fold cross validation [23] is used to evaluate the performance of our proposed tool. We randomly divide the dataset into 10 folds. Of these 10 folds, 9 folds are use to train a statistical model, while the last one fold is used to evaluate the performance of the model. We iterate the whole process 10 times, and record the average performance across the 10 iterations. The distribution of labels in the training and test folds are the same as the original dataset to simulate the actual usage of our tool. Stratified cross validation is a standard evaluation setting, which is widely used in software engineering studies, c.f., [26], [27], [28], [29], [30], [31].

We use the implementation of the 2 feature selection techniques and the 5 classification techniques in Weka [32].[9] By default, we select as features terms whose feature selection scores are in the top 10% ranked list. For kNN, we set the number of neighbors to 5. For the other classification techniques, we use their default settings in Weka.

B. Evaluation Metrics

To evaluate the predictive performance of our proposed tool, we create a confusion matrix to store prediction results. Table II presents an example of a confusion matrix. The rows of the matrix correspond to predicted labels of bug reports. The columns of the matrix correspond to correct labels of bug reports. A cell in the matrix contains the number of bug reports of a particular predicted label and a particular correct label.

For each bug report, there would be 4 possible outcomes: a bug report can be classified as a configuration bug report when it truly is a configuration bug report (true positive, TP); it can be classified as a configuration bug report when actually it is a non-configuration bug report (false positive, FP); it can be classified as a non-configuration bug report when it is actually a configuration bug report (false negative, FN); or it can be

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

TABLE II. CONFUSION MATRIX.

Classified as	True Class	
	Configuration	Non-configuration
Configuration	TP	FP
Non-configuration	FN	TN

classified as a non-configuration bug report and it truly is a non-configuration bug report (true negative, TN). By using the values stored in the confusion matrix, in this paper, we calculate the accuracy, precision, recall and F-measure scores for each label (i.e., configuration and non-configuration) to evaluate the performance of our proposed tool.

- **Accuracy**: the number of correctly classified bugs (both configuration and non-configuration bugs) over the total number of bugs, i.e., $Acc = \frac{TP+TN}{TP+FP+TN+FN}$.
- **Configuration Precision**: the proportion of bugs that are correctly labeled as configuration bugs among those labeled as configuration bugs, i.e., $P(C) = \frac{TP}{TP+FP}$.
- **Configuration Recall**: the proportion of configuration bugs that are correctly labeled, i.e., $R(C) = \frac{TP}{TP+FN}$.
- **Non-configuration Precision**: the proportion of bugs that are correctly labeled as non-configuration bugs among those labeled as non-configuration bugs, i.e., $P(NC) = \frac{TN}{TN+FN}$.
- **Non-configuration Recall**: the proportion of non-configuration bugs that are correctly labeled, i.e., $R(NC) = \frac{TN}{TN+FP}$.
- **F-measure**: a summary measure that combines both precision and recall – it evaluates if an increase in precision (recall) outweighs a reduction in recall (precision). For configuration F-measure, it is $F(C) = \frac{2*P(C)*R(C)}{P(C)+R(C)}$. And for non-configuration F-measure, it is $F(NC) = \frac{2*P(NC)*R(NC)}{P(NC)+R(NC)}$.

Notice that precision and recall are both important metrics for configuration bug report prediction since they measure quality of our tool in two aspects. If the precision is low, then the developer would not use the tool, due to a high number of false positives. If the recall is low, developers would not use the tool also, since most configuration (non-configuration) bug reports are not successfully predicted. There is a trade off between precision and recall, and one can increase precision by sacrificing recall (and vice versa) [23]. F-measure, which is the harmonic mean of precision and recall, is often used to judge whether an increase in precision outweighs a loss in recall (and vice versa) [23]. In many past papers in software engineering literature, e.g., [33], [34], [35], [29], F-measure is often used as a summary measure. Thus, in this paper, we choose configuration F-measure and non-configuration F-measure as two most important evaluation metrics.

C. Research Questions

We are interested to answer the following research questions:

RQ1 *How effective is our proposed tool? Which feature selection and classification techniques achieves the best performance? How much improvement could our proposed tool achieve over the baseline method proposed by Arshad et al.?*

In our proposed tool, we propose the usage of 2 feature selection techniques and 5 classification techniques. Thus, in total, there are 10 different combinations of feature selection and classification techniques. We would like to investigate which combination of these 2 techniques could achieve the best performance. Also, Arshad et al. have proposed a keyword query based approach to identify configuration bugs [19]. Thus, we consider their approach as a baseline method, and compare our proposed approach with theirs.

Answer to this research question would shed light to whether our proposed tool advances the state-of-the art methods. To answer this research question, we first investigate which combination of our proposed method achieves the best performance, and then we compare this combination with the baseline method proposed by Arshad et al.

RQ2 *Can feature selection techniques really improve the performance of our tool?*

Since we apply feature selection techniques to preprocess bug reports, and then use the selected features (i.e., terms) to train a classifier, we also investigate whether feature selection techniques really improve the performance of our tool. To answer this research question, we compare the performance of our tool with and without feature selection techniques.

RQ3 *Do different numbers of selected features affect the performance of our proposed tool?*

By default, we select the top 10% features with the highest feature selection scores. We investigate whether different number of selected features (i.e., terms) would affect the performance of our proposed tool. To answer this research question, we vary the number of selected features from 1% to 20% of the total number of features.

RQ4 *What are the best features for discriminating whether a bug report is a configuration bug report or not?*

Aside from producing a model that can identify configuration bug reports, we are also interested in finding discriminative features that could help in distinguishing configuration bug reports and non-configuration bug reports. To answer this research question, we compute the information gain scores of all the features that we collected and present the top features.

D. RQ1: Performance of Our Tool

Tables III and IV present the accuracy, configuration F-measure, and non-configuration F-measure scores for the 5 datasets and 5 classification algorithms when information gain (IG) and chi-square (CHI) are used as feature selection techniques, respectively. We notice that there are only small differences between scores corresponding to these two feature selection techniques. For example, the accuracy, configuration F-measure, and non-configuration F-measure for IG with kNN are 0.813, 0.229, and 0.888 respectively, while these values are 0.814, 0.224, and 0.889 for CHI with kNN. In the following

TABLE III. ACCURACY, CONFIGURATION F-MEASURE (CONF. F-MEASURE), AND NON-CONFIGURATION F-MEASURE (NON. F-MEASURE) FOR THE 5 DATASETS WITH INFORMATION GAIN AS THE FEATURE SELECTION TECHNIQUE. THE LAST COLUMN SHOWS THE AVERAGE ACCURACY AND F-MEASURE SCORES ACROSS THE 5 DATASETS.

Evaluation	Techniques	accumulo	activemq	camel	flume	wicker	Average
Accuracy	NB	0.823	0.823	0.740	0.681	0.931	0.800
	NBM	0.840	0.823	0.734	0.703	0.954	0.811
	kNN	0.823	0.829	0.728	0.717	0.968	0.813
	SVM	0.829	0.840	0.733	0.728	0.964	**0.819**
	BN	0.779	0.846	0.750	0.692	0.964	0.806
Conf. F-measure	NB	0.500	0.340	0.418	0.341	0.307	0.381
	NBM	0.525	0.392	0.463	0.450	0.422	**0.450**
	kNN	0.238	0.167	0.331	0.288	0.120	0.229
	SVM	0.340	0.125	0.236	0.333	0.074	0.222
	BN	0.167	0.308	0.410	0.000	0.039	0.185
Non. F-measure	NB	0.893	0.898	0.833	0.790	0.964	0.875
	NBM	0.904	0.896	0.823	0.797	0.976	0.880
	kNN	0.900	0.904	0.829	0.823	0.984	0.888
	SVM	0.902	0.912	0.839	0.829	0.982	**0.892**
	BN	0.873	0.913	0.842	0.818	0.982	0.885

TABLE IV. ACCURACY, CONFIGURATION F-MEASURE (CONF. F-MEASURE), AND NON-CONFIGURATION F-MEASURE (NON. F-MEASURE) FOR THE 5 DATASETS WITH CHI-SQUARE AS THE FEATURE SELECTION TECHNIQUE.

Evaluation	Techniques	accumulo	activemq	camel	flume	wicker	Average
Accuracy	NB	0.823	0.823	0.738	0.681	0.930	0.799
	NBM	0.845	0.817	0.738	0.703	0.953	0.811
	kNN	0.823	0.823	0.737	0.717	0.968	0.814
	SVM	0.834	0.834	0.733	0.728	0.964	**0.819**
	BN	0.779	0.846	0.750	0.692	0.964	0.806
Conf. F-measure	NB	0.500	0.340	0.421	0.341	0.304	0.381
	NBM	0.533	0.385	0.467	0.450	0.414	**0.450**
	kNN	0.238	0.114	0.360	0.288	0.120	0.224
	SVM	0.348	0.121	0.240	0.333	0.074	0.223
	BN	0.167	0.308	0.410	0.000	0.039	0.185
Non. F-measure	NB	0.893	0.898	0.831	0.790	0.963	0.875
	NBM	0.907	0.893	0.827	0.796	0.975	0.880
	kNN	0.900	0.902	0.834	0.823	0.984	0.889
	SVM	0.905	0.909	0.838	0.829	0.982	**0.892**
	BN	0.873	0.913	0.842	0.818	0.982	0.806

research questions, we would use information gain as the default feature selection technique, since the difference between these 2 techniques are slight.

In terms of accuracy and non-configuration F-measure scores, the differences among different classification techniques are small. For example, in Table III, the highest accuracy is 0.819 (SVM), and the lowest accuracy is 0.800 (naive Bayes (NB)) – SVM only improves NB by 2.4%. However, for configuration F-measure, the differences among different classification techniques are high. For example, in Table III, the highest configuration F-measure is 0.450 (naive Bayes multinomial (NBM)), and the lowest configuration F-measure is 0.185 (Bayesian network (BN)) – NBM improves BN by 143.2%. Considering that configuration bug reports are only a small proportion of the total bug reports, it is much harder to predict configuration bug report correctly. For this reason, naive Bayes multinomial (NBM) achieves the best performance since it achieves similar accuracy and non-configuration F-measure values, but a much better configuration F-measure value.

Tables V present the experiment results of our proposed tool (IG with NBM) compared with Arshad et al.'s method. The accuracy, configuration F-measure, and non-configuration F-measure of our proposed tool vary from 0.703 - 0.954, 0.392 - 0.525, and 0.798 - 0.976. On average across the 5 datasets, our proposed tool outperforms Arshad et al.'s method by 8.34%, 103.70%, and 4.24% in terms of accuracy, configuration F-measure, and non-configuration F-measure. Notice that the average configuration F-measure for Arshad et al.'s method is 0.221, which means that their method can not identify configuration bug reports well.

E. RQ2: Benefit of Feature Selection

Table VI present the experiment results of directly using classification techniques without feature selection. We notice

TABLE V. OUR PROPOSED TOOL (INFORMATION GAIN WITH NAIVE BAYES MULTINOMIAL) VS. ARSHAD ET AL.'S METHOD.

Evaluation	Techniques	accumulo	activemq	camel	flume	wicker	Average.
Accuracy	Ours	0.840	0.823	0.734	0.703	0.954	0.811
	Arshad et al.	0.807	0.737	0.651	0.681	0.866	0.748
	Improv.	**4.11%**	**11.63%**	**12.79%**	**3.16%**	**10.22%**	**8.34%**
Conf. F-measure	Ours	0.525	0.392	0.463	0.450	0.422	0.450
	Arshad et al.	0.364	0.080	0.271	0.360	0.031	0.221
	Improv.	**44.26%**	**390.20%**	**70.92%**	**25.19%**	**1243%**	**103.70%**
Non. F-measure	Ours	0.904	0.896	0.823	0.798	0.976	0.880
	Arshad et al.	0.886	0.847	0.771	0.788	0.928	0.844
	Improv.	**1.99%**	**5.86%**	**6.86%**	**1.32%**	**5.20%**	**4.24%**

TABLE VI. ACCURACY, CONFIGURATION F-MEASURE (CONF. F-MEASURE), AND NON-CONFIGURATION F-MEASURE (NON. F-MEASURE) FOR THE 5 DATASETS WITHOUT FEATURE SELECTION.

Evaluation	Techniques	accumulo	activemq	camel	flume	wicker	Average
Accuracy	NB	0.641	0.674	0.693	0.663	0.888	0.712
	NBM	0.829	0.749	0.754	0.695	0.945	0.794
	kNN	0.840	0.817	0.722	0.713	0.966	0.812
	SVM	0.834	0.811	0.739	0.706	0.965	0.811
	BN	0.779	0.846	0.750	0.692	0.964	0.806
Conf. F-measure	NB	0.381	0.387	0.412	0.420	0.251	0.370
	NBM	0.608	0.450	0.499	0.541	0.429	0.505
	kNN	0.256	0.000	0.191	0.259	0.000	0.141
	SVM	0.423	0.298	0.436	0.423	0.333	0.383
	BN	0.167	0.308	0.410	0.000	0.039	0.185
Non. F-measure	NB	0.747	0.778	0.792	0.763	0.939	0.804
	NBM	0.890	0.837	0.837	0.772	0.969	0.861
	kNN	0.910	0.899	0.832	0.822	0.983	0.889
	SVM	0.903	0.891	0.830	0.803	0.982	0.882
	BN	0.873	0.913	0.842	0.818	0.982	0.885

that in general feature selection is useful. For example, on average, across the 6 datasets, the accuracy, configuration F-measure, and non-configuration F-measure scores for naive Bayes with IG are 0.800, 0.381, and 0.875, respectively, while these values for naive Bayes are 0.712, 0.370, and 0.804, respectively. For NBM and SVM with IG, we notice that their respective configuration F-measures are lower than those of NBM and SVM without feature selection. However, considering their accuracy and non-configuration F-measures, there are still some improvements when feature selection is used.

F. RQ3: Effect of Varying the Number of Selected Features

We investigate the effect of varying the numbers of selected features on the performance of our proposed tool. We vary the number of features selected from 1% to 20% of the total number of terms. We plot the resultant accuracy, configuration F-measure, and non-configuration F-measure scores for Camel in Figure 4. The results show that when we select more than 3% of the total number of features, the performance of our proposed tool is stable with various numbers of selected features. Due to space limitations, we omit the figures for accumulo, activemq, flume, and wicket datasets, however, we confirm that for these projects, the trends are the same as camel.

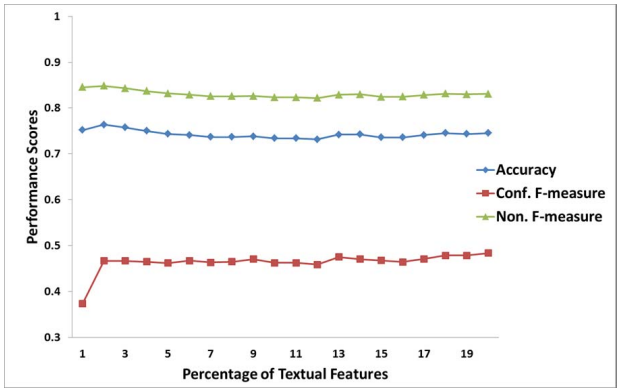

Fig. 4. Experiment Results of Our Proposed Tool on Camel with Number of Textual Features Varying from 1% to 20% of The Total Number of Terms.

G. RQ4: Best Features

We extract thousands of features corresponding to the number of times terms appear in the bug reports. We report the top 10 features sorted based on their information gain scores in Table VII. We notice that the information gain score is low (the highest possible value would be 1), which represents that one feature alone is not sufficient to discriminate configuration

TABLE VII. TOP-10 MOST DISCRIMINATIVE FEATURES BASED ON INFORMATION GAIN (IG) SCORES.

accumulo		activemq		camel		flume		wicker	
plugin	0.065	mav	0.087	spring	0.034	depens	0.039	propers	0.018
gener	0.064	plugin	0.077	karaf	0.028	mav	0.039	depens	0.015
workspac	0.057	build	0.055	install	0.027	build	0.038	artifactid	0.012
mav	0.056	thread	0.053	pack	0.025	point	0.029	method	0.012
nofollow	0.056	problem	0.050	vers	0.025	inclus	0.029	main	0.011
extern	0.056	snapshot	0.046	bundl	0.024	master	0.028	pack	0.010
depens	0.052	project	0.046	featur	0.022	fileconfig	0.025	pag	0.009
trunk	0.052	upd	0.037	osg	0.018	config	0.025	fil	0.009
config	0.050	vers	0.037	mav	0.018	status	0.020	mav	0.008
http	0.048	localhost	0.032	camel	0.018	plugin	0.020	quickstatrt	0.007

bug reports from non-configuration bug reports. The keywork "config" which is used by Arshad et al. to identify configuration bug reports, appears in the top 10 most discriminative features of accumulo and flume. As all of the 5 projects are using maven ("mav") as a build tool, it appears in the top 10 most discriminative features of all of the 5 projects. Note that "mav" is the stemmed word of maven. Some other features, such as "depens" (dependency), "build", "propers" (property), "pack" (package), "fil" (file) are all good indicators to identify whether a bug report is a configuration bug report.

H. Threats to Validity

Threats to internal validity relate to errors in our experiments. We have double checked our experiments and the datasets collected from the 5 projects, and we have also manually checked the bug reports to ensure that they are configuration bug reports, still there could be errors that we do not notice.

Threats to external validity relate to the generalizability of our results. We have analyzed 3,203 bug reports from 5 open source software projects. In the future, we plan to reduce this threat further by analyzing more bug reports from open source and commercial software projects .

Threats to construct validity refer to the suitability of our evaluation metrics. We use accuracy, configuration, and non-configuration F-measure scores as the main evaluation metrics which are also used by past software engineering studies to evaluate the effectiveness of a prediction technique [33], [34], [35], [29], [18]. Thus, we believe there is little threat to construct validity.

VII. RELATED WORK

There have been a number of studies on configuration management [9], [19], [11], [12], [36], [37]. Yin et al. perform an empirical study on configuration issues in one commercial and 4 open source software systems [9]. They conclude that a majority of configuration bugs are due to wrong parameter setting. Arshad et al. extract configuration bugs from GlassFish and JBoss, and they characterize the configuration bugs from several dimensions, i.e., problem-type, problem-time, problem-manifestation, and problem-culprit [19]. Based on their findings, they also develop a tool named ConfGauge which injects parameter-based configuration issues into software systems. Zhang and Ernst combines static analysis, dynamic profiling, and statistical analysis to detect problems in configuration files [11]. Xu et al. propose SPEX to automatically infer configuration constraints from source code, and then use these constraints to expose misconfiguration vulnerabilities, and detect error-prone configuration design [12]. Attariyan and Flinn propose AutoBash which diagnoses configuration errors by considering the causal dependencies of test case executions to detect similarities between a sick computer and a reference computer [36]. Later, Attariyan and Flinn develop ConfAid which traces configuration errors by leveraging data flow analysis [37]. Our study complements the above studies; we predict whether a bug report is a configuration bug or not to help developers reduce debugging effort.

There have been a number of studies on characterizing or predicting the types of bugs [38], [39], [40], [41]. Gegick et al. propose the usage of text mining techniques to identify whether a bug is a security bug or not [38]. Zaman et al. perform an empirical study on security bugs and performance bugs in Firefox [39]. They find security bugs need more time to be fixed, while performance bugs are not that different from other bugs, in terms of bug fix time, but more files need to be changed to fix them. Xia et al. perform an empirical study on bugs in software build systems such as Ant, Maven, CMake and QMake, and they find that 21.35% of the build system bugs are related to external interface problems [40]. Thung et al. propose a method to automatically categorize bug reports into two families: control and data flow, and structural [41]. Xia et al. propose a fuzzy-set based feature selection approach to categorize defect based on its fault triggering conditions [42]. Our work complements the above studies; we classify a bug as a configuration or non-configuration bug.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we propose an automated tool which applies feature selection and classification techniques to build a statistical model from the natural-language description of historical bug reports, to predict whether a newly submitted bug report is a configuration bug or not. Various feature selection techniques (e.g., information gain and Chi-square) and various classification techniques (e.g., naive Bayes, naive Bayes multinomial, kNN, SVM, and Bayesian network) have been investigated. We evaluate our proposed tool on 5 open source projects including a total of 3,203 bug reports. The experiment results show naive Bayes multinomial with information gain performs the best; on average across the 5 projects, its accuracy, configuration F-measure and non-configuration F-measure are 0.811, 0.450, and 0.880, respectively, which improve Arshad et al.'s method

by 8.34%, 103.7%, and 4.24%, respectively.

In the future, we plan to improve the effectiveness of our proposed tool further. We also plan to experiment with even more bug reports from more projects.

ACKNOWLEDGMENT

This research is sponsored in part by NSFC Program (No.61103032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (2014BAH24F02).

REFERENCES

[1] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do internet services fail, and what can be done about it?" in *USENIX Symposium on Internet Technologies and Systems*, vol. 67. Seattle, WA, 2003.

[2] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[3] R. Johnson, "More details on todays outage," *http://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919*, 2010.

[4] Y. Sverdlik., "Microsoft: Misconfigured network device led to azure outage," *http://www.datacenterdynamics.com/focus/archive/2012/07/microsoft-misconfigured-network-device-led-azure-outage*, 2012.

[5] A. Team, "Summary of the amazon ec2 and amazon rds service disruption in the us east region," *http://aws.amazon.com/cn/message/65648/*, 2011.

[6] N. Eddy, "Human error caused google glitch," *http://www.eweek.com/c/a/Enterprise-Applications/Human-Error-Caused-Google-Glitch/*, 2009.

[7] B. Hale, "Why every it practitioner should care about network change and configuration management," *http://web.swcdn.net/creative/pdf/Whitepapers/Why_Every_IT_Practitioner_Should_Care_About_NCCM.pdf*, 2012.

[8] A. Kapoor, "Web-to-host: Reducing total cost of ownership," Technical Report 200503, The Tolly Group, Tech. Rep., 2000.

[9] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 159–172.

[10] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure." in *OSDI*, vol. 4, 2004, pp. 245–257.

[11] S. Zhang and M. D. Ernst, "Automated diagnosis of software configuration errors," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 312–321.

[12] T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy, "Do not blame users for misconfigurations," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 244–259.

[13] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *ICML*, vol. 97, 1997, pp. 412–420.

[14] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[15] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer, 2012.

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

[17] T.-D. B. Le and D. Lo, "Will fault localization work for these failures? an automated approach to predict effectiveness of fault localization tools," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013, pp. 310–319.

[18] X. Xia, D. Lo, X. Wang, X. Yang, S. Li, and J. Sun, "A comparative study of supervised learning algorithms for re-opened bug prediction," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*. IEEE, 2013, pp. 331–334.

[19] F. A. Arshad, R. J. Krause, and S. Bagchi, "Characterizing configuration problems in java ee application servers: An empirical study with glassfish and jboss," pp. 198–207, 2013.

[20] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*. ACM press New York, 1999, vol. 463.

[21] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[22] A. McCallum, K. Nigam *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752. Citeseer, 1998, pp. 41–48.

[23] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

[24] D. Kollar and N. Friedman, *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[25] D. Tutorial, "The word vector tool."

[26] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," in *Reverse Engineering (WCRE), 2010 17th Working Conference on*. IEEE, 2010, pp. 249–258.

[27] X. Xia, Y. Feng, D. Lo, Z. Chen, and X. Wang, "Towards more accurate multi-label software behavior learning," in *CSMR-WCRE*, 2014.

[28] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Proceedings of the Tenth International Workshop on Mining Software Repositories*. IEEE Press, 2013, pp. 287–296.

[29] Y. Tian, J. Lawall, and D. Lo, "Identifying linux bug fixing patches," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 386–396.

[30] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. IEEE, 2008, pp. 346–355.

[31] F. Thung, D. Lo, and J. L. Lawall, "Automated library recommendation," in *WCRE*, 2013, pp. 182–191.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[33] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 63.

[34] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes." in *SIGSOFT FSE*, 2011, pp. 15–25.

[35] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 382–391.

[36] M. Attariyan and J. Flinn, "Using causality to diagnose configuration bugs." in *USENIX Annual Technical Conference*, 2008, pp. 281–286.

[37] ——, "Automating configuration troubleshooting with dynamic information flow analysis." in *OSDI*, 2010, pp. 237–250.

[38] M. Gegick, P. Rotella, and T. Xie, "Identifying security bug reports via text mining: An industrial case study," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 2010, pp. 11–20.

[39] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on firefox," in *Proceedings of the 8th working conference on mining software repositories*. ACM, 2011, pp. 93–102.

[40] X. Xia, X. Zhou, D. Lo, and X. Zhao, "An empirical study of bugs in software build systems," in *Quality Software (QSIC), 2013 13th International Conference on*. IEEE, 2013, pp. 200–203.

[41] F. Thung, D. Lo, and L. Jiang, "Automatic defect categorization," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 2012, pp. 205–214.

[42] X. Xia, D. Lo, X. Wang, and B. Zhou, "Automatic defect categorization based on fault triggering conditions," in *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*. IEEE, 2014.