Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-aware Perspective

Yaming Tang Fei Zhao Yibiao Yang Hongmin Lu Yuming Zhou Baowen Xu
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
tangyaming_nju@sina.com njumrz@gmail.com yangyibiao@smail.nju.edu.cn
hmlu, zhouyuming, bwxu@nju.edu.cn

***Abstract*—In order to identify vulnerable software components, developers can take software metrics as predictors or use text mining techniques to build vulnerability prediction models. A recent study reported that text mining based models have higher recall than software metrics based models. However, this conclusion was drawn without considering the sizes of individual components which affects the code inspection effort to determine whether a component is vulnerable. In this paper, we investigate the predictive power of these two kinds of prediction models in the context of effort-aware vulnerability prediction. To this end, we use the same data sets, containing 223 vulnerabilities found in three web applications, to build vulnerability prediction models. The experimental results show that: (1) in the context of effort-aware ranking scenario, text mining based models only slightly outperform software metrics based models; (2) in the context of effort-aware classification scenario, text mining based models perform similarly to software metrics based models in most cases; and (3) most of the effect sizes (i.e. the magnitude of the differences) between these two kinds of models are trivial. These results suggest that, from the viewpoint of practical application, software metrics based models are comparable to text mining based models. Therefore, for developers, software metrics based models are practical choices for vulnerability prediction, as the cost to build and apply these models is much lower.**

Keywords-software metrics; text mining; vulnerability; prediction; effort-aware

I. INTRODUCTION

During software development process, it is hard to develop reliable software at a time. Developers often have to test software components to ensure that software components have no security vulnerabilities. However, in practice, test resources such as testers and time are limited. Consequently, developers should focus on those components which are most likely to contain vulnerability. In this context, developers could take advantage of prediction models to help reduce the effort needed to mitigate vulnerabilities.

In the last decades, much effort was devoted to build and apply models to predict software defects. While vulnerabilities are a subset of software defects, they deserve special attention because of the following special features. First, there are many more software defects than vulnerabilities in software components [2]. Second, testers should be familiar with both the software architecture and the attacker's mindset in order to evaluate software security adequately [21]. Third, users will report defects to developers while hackers will limit the leakage and dissemination of vulnerabilities [2]. In addition, people argue about how much information to disclose about vulnerabilities among security community [22]. Due to the above-mentioned reasons, publicly available vulnerability data are much less compared to defect data. Consequently, in the literature, vulnerability prediction studies are scarce.

In [1], Scandariato et al. used text mining and machine learning techniques to build software vulnerability prediction models to predict vulnerable components. Their results showed that text mining based models could achieve a good classification performance. In [2], Walden et al. further investigated whether text mining based models was superior to software metrics based models when used to predict vulnerable components. Their experimental results showed that text mining based models have a higher recall than software metrics based models.

However, these previous studies did not take into account the code inspection effort to determine whether a component was vulnerable. Indeed, when developers inspect their codes to determine whether there are security vulnerabilities, they will spend more effort on larger modules. When applying the prediction models in practice, we hence should consider the influence of module size on vulnerability prediction. In other words, we should use effort-aware performance indicators to evaluate the prediction effectiveness of vulnerability prediction models. Currently, it is unclear whether text mining based models is still superior to software metrics based models under effort-aware evaluation.

In this study, we aim to compare the predictive power of text mining based models and software metrics based models in the context of effort-aware vulnerability prediction. As in [2], the subject systems used in our study consist of three web applications containing 223 vulnerabilities, including Drupal, Moodle and PHPMyAdmin. Based on the data sets collected by Walden et al. in the literature [2], we first build text mining based models and software metrics based models. After that, we evaluate the models by comparing them to each other to determine how effective they are. In order to obtain comprehensive and realistic performance evaluations, we evaluate the prediction performance of these models with respect to both ranking and classification scenarios under cross-validation and across-project prediction.

978-1-4673-7989-2/15 $31.00 © 2015 IEEE
DOI 10.1109/QRS.2015.15

IEEE computer society

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

Our experimental results show that: (1) in the context of effort-aware ranking scenario, text mining based models only slightly outperform software metrics based models; (2) in the context of effort-aware classification scenario, text mining based models perform similarly to software metrics based models in most cases; and (3) most of the effect sizes (i.e. the magnitude of the differences) between these two kinds of models are trivial. These results suggest that, from the viewpoint of practical application, software metrics based models are comparable to text mining based models. Therefore, for developers, software metrics based models are practical choices for vulnerability prediction, as the cost to build and apply the models is much lower.

The rest of the paper is organized as follows. Section II introduces the background on software vulnerability prediction and effort-aware prediction models. Section III describes the employed experimental methodology, including the research questions to be answered, the independent and dependent variables, the effort-aware software vulnerability prediction models under study, the performance indicators, and the data analysis method. Section IV reports the experimental results. Section V concludes the paper and outlines the direction for future work.

II. BACKGROUND

In this section, we introduce the background on vulnerability prediction and effort-aware prediction. Due to limited space, only the most related work in the field will be introduced.

A. Vulnerability Prediction Models

The purpose of building prediction models is to determine whether software components (source code files, classes, functions, etc.) are vulnerable or not. According to [1], prediction models can be described in five dimensions, including the type of prediction features, the source of the vulnerability data, the type of prediction techniques, the applications used for the validation, and the performance indicators. In our work, we focus on the type of prediction features used and the type of prediction techniques.

For the type of prediction features, our choices are divided into two categories: software metrics and text mining. Shin et al. explored whether complexity metrics could be used to build models to locate security issues [23, 24]. Chowdhury et al. explored whether complexity, coupling, and cohesion metrics could be used to build vulnerability prediction models [25]. Zimmermann et al. explored whether there is a correlation between vulnerabilities and various metrics, including code churn, code complexity, and dependencies [26]. Hata et al. [27] and Mizuno et al. [28] applied text features to predict defects in software. Aversano et al. [15] treated the source code of the changes as text to build prediction models to explore whether the introduced changes were buggy or not.

For the type of prediction techniques, with the help of a software engineering expert, Zhong et al. [3] used several clustering techniques to build defect classification models, including K-Means and Neural-Gas. Zhao and Zhou [4], [5] used several clustering techniques to build unsupervised models to rank module defect-proneness, including K-Means and X-Means. Scandariato et al. [1], [2] used classification algorithms in machine learning techniques, including *Naive Bayes* and *Random Forest*, to build vulnerability prediction models.

B. Effort-aware Prediction Models

Vulnerability prediction models mentioned above treat each software component equally, regardless of their respective size. In practice, however, it is more important to know the effort-aware prediction performance of vulnerability prediction models. In the last decade, many researchers investigated whether there is a relationship between fault-proneness and module size.

Koru et al. [6]–[9] suggested that we should first inspect smaller modules to detect more faults per unit code inspection effort. More specifically, they suggested ranking the modules in ascending order by module size for code inspection. The reason was that the relationship between module size and the number of faults was found logarithmic [7], [8], indicating that fault-proneness increased as module size increased at a slower rate. Menzies et al. [10] named the method of "smaller modules inspected first" with ManualUp model. In particular, they found that the ManualUp model had a good prediction performance in the context of effort-aware fault prediction. More recently, Zhou et al. [11] found that the ManualUp model was even competitive to the logistic regression model. All these studies show that the module size should be taken into consideration when developers inspect source code of system with many components.

III. RESEARCH METHODOLOGY

In this section, we first formulate the three research questions that will be answered. Then, we describe the data sets used in our study. Next, we describe the investigated independent and dependent variables. After that, we present the effort-aware software vulnerability prediction models under study. Furthermore, we introduce the performance indicators for evaluating the effectiveness of vulnerability prediction models in the context of effort-aware vulnerability prediction. Finally, we give the data analysis method.

A. Research Questions

Our study aims to compare the effort-aware prediction effectiveness between text mining based models and software metrics based models. In other words, we want to investigate the prediction effectiveness of the two types of models proposed by Walden et al. in [2] in the context of effort-aware software vulnerability prediction. Walden et al. [2] have revealed that text mining based models outperformed software metrics based models in the context of non-effort-aware software vulnerability prediction. If the experimental results show that text mining based models outperform software metrics based models only slightly, then it would be a good choice for practitioners to apply software metrics based models to predict vulnerable software components in practice. The reason is that the cost to build and apply software metrics based models is much lower than text mining based models. In order to determine the practical value of the two types of models, our study will investigate the following three research questions:

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

- **RQ 1:** *In the ranking scenario, do text mining based models outperform software metrics based models in the context of effort-aware vulnerability prediction?*
- **RQ 2:** *In the classification scenario, do text mining based models outperform software metrics based models in the context of effort-aware vulnerability prediction?*
- **RQ 3**: *Are text mining based models more cost-effective than software metrics based models in effort-aware vulnerability prediction?*

The purpose of RQ1 and RQ2 is to determine whether text mining based models outperform software metrics based models in the context of effort-aware vulnerability prediction in both the classification scenario and ranking scenario or not. The purpose of RQ3 is to investigate whether text mining based models are more cost-effective when taking into account the cost to build and apply the models.

B. Data Sets

In this study, we use the same projects investigated in James Walden et al.'s study [2]. These projects are the following three open-source web applications written in PHP: Drupal, Moodle, and PHPMyAdmin. The data sets containing a total of 223 vulnerabilities from multiple versions of these three applications were collected by Walden et al. Table I summaries the three data sets used in this study. The first column is the system name. The second to the fifth columns respectively reports for each system the total number of files, the total number of vulnerable file, the percentage of vulnerable files, and the number of text features extracted from each system. As can be seen, the percentage of vulnerable files varied a lot. A large percentage of files in Drupal were vulnerable while only an extremely small percentage of files in Moodle were vulnerable.

C. Depedent and Independent Variables

In [2], Walden et al. compared text mining based models with software metrics based models in the context of non-effort-aware vulnerability prediction. In this study, we aim to investigate the prediction effectiveness of both models in the context of effort-aware vulnerability prediction. Therefore, the dependent variable used in our study is the same as that in [2]. More specifically, a source code file was labeled vulnerable if there was at least one vulnerability and otherwise was labeled clean. According to this criterion, the dependent variable in our study is a binary variable which were labeled as "yes" or "no".

The independent variables used in our study are also the same as those in [2]. Table II summarizes the twelve software metrics used in this study, including the metrics name and the description. For the part of text mining, every PHP file was tokenized by the PHP's built-in token_get_all function. During this process, every source file was represented as a list of tokens and their corresponding frequencies without comments and whitespace. In addition, the function converted the string and numeric literals into fixed tokens, for instance, T_STRING was used to replace the content of a string. Finally, every source file was represented as a term vector when building prediction models as predictors.

TABLE I. DESCRIPTIVED STATISTICS FOR THE APPLICATIONS

System	# files	# Vulnerable files	Vulnerable files%	#Text features
Drupal	202	62	30.68	3886
PHPMyAdmin	322	27	8.39	5232
Moodle	2942	24	0.82	18306

TABLE II. SUMMARIZATION OF SOFTWARE METRICS

Name	Description
Lines of code	Number of lines in a source file, excluding lines without PHP tokens
Lines of code (non-HTML)	Same as Lines of code, except HTML content embedded
Number of functions	Number of function and method definitions in a file
Cyclomatic complexity	The size of a control flow graph after linear chains of nodes are collapsed into one
Maximum nesting complexity	The maximum depth to which loops and control structures in the file are nested
Halstead's volume	A volume estimate ((N1 +N2) log n1 + n2) using the number of unique operators (n1) and operands (n2) and the number of total operators (N1) and operands (N2) in the file
Total external calls	The number of instances where a statement in the file being measured invokes a function or method defined in a different file
Fan-in	The number of files (excluding the file being measured) which contain statements that invoke a function or method defined in the file being measured
Fan-out	The number of files (excluding the file being measured) containing functions or methods invoked by statements in the file being measured
Internal functions or methods called	The number of functions or methods defined in the file being measured which are called at least once by a statement in the same file
External functions or methods called	The number of functions or methods defined in other files which are called at least once by a statement in the file being measured
External calls to functions or methods	The number of files (excluding the file being measured) calling a particular function or method defined in the file being measured, summed across all functions and methods in the file being measured

D. Effort-aware Vulnerability Prediction Models

In our study, we leverage both the metrics and the term vector as predictors to build prediction models. In line with previous work [2], we do not apply feature selection in building prediction models. In particular, we also use *Random Forest* as our primary machine learning algorithm with the same configuration of Weka 3.7 and the size of the forest set to 100 trees.

Recently, Y. Yang et al. used the prediction models to compute the predicted fault-proneness per *SLOC* and found that the resulting models were far better than the random model in effort-aware defection prediction [12]. In this study, we use Pr/SLOC (rather than Pr) to rank files, where Pr is the probability of a file being vulnerable predicted using the "T" model (i.e. text mining based model in [2]) or the "M" model (i.e. software metrics based model in [2]). For simplicity of presentation, we use T' and M' to respectively denote the improved T model and the improved M model.

For the individual application, we apply the two prediction models to the same test set to give every file a predictive value

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

as its predictive vulnerability-proneness (i.e. T model and M model). After get the predictive value of every file in the test set, we divide it by the corresponding module size (represented by the lines of code (non-HTML)) to get the new predictive value as its vulnerability-proneness (i.e. T' model and M' model).

E. Performance Indicators

We evaluate the effectiveness of a given vulnerability prediction model under the following two typical application scenarios: ranking and classification. In the ranking scenario, software components are ranked in descending order by the degree of their predicted vulnerability-proneness. Developers can easily select as many vulnerability-prone software components from the ranking list for inspecting as available resources will allow. In the classification scenario, software components are first classified into the vulnerability-prone category and the not-vulnerability-prone category according to their predictive vulnerability-proneness. Then, software components which are predicted as vulnerability-prone will be targeted for code inspection. In both scenarios, when evaluating the predictive effectiveness of a software vulnerability prediction model, we take into consideration the effort required to inspect those components predicted as vulnerability-prone ones to determine whether they are real vulnerable components. We use the lines of code without HTML contents as a proxy of the effort required to inspect the suspected component. We describe the effort-aware predictive performance evaluation indicators with respect to both ranking and classification scenarios respectively as follows.

1) Effort-aware classification performance evaluation

We evaluate the effort-aware classification effectiveness of a software vulnerability prediction model with the measure of ER [11]. The ER measure indicates the proportion of the reduced lines of code to be inspected by a model *m* compared with the random model that achieves the same recall of vulnerable components. Assume that the application under our study consists of n components. For $1 \le i \le n$, let s_i be the lines of code of the component and f_i be a binary variable indicating whether component i is vulnerable or not (i.e. $f_i = 1$ if component i is vulnerable and otherwise $f_i = 0$). In addition, for $1 \le i \le n$, let p_i be 1 if component i is predicted by the model m as vulnerability-prone and 0 otherwise. In the classification scenario, only those software components predicted to be vulnerability-prone will be tested or inspected. In this context, the effort-aware classification effectiveness of the prediction model m can be formally defined as follows:

$$ER(\mathrm{m}) = \frac{Effort(random\ model) - Effort(m)}{Effort(random\ model)}$$

In the formula above, $Effort(m)$ is the ratio of the total lines of code in those predicted vulnerable components to the total lines of code in the system, i.e.

$$Effort(m) = \sum_{i=1}^{n} s_i \times p_i \Big/ \sum_{i}^{n} s_i$$

$Effort(random\ model)$ is the ratio of the lines of code needed to inspect by a random selection model to the total lines of code under the condition that the random model achieves the same recall of vulnerable components as the prediction model m, i.e.

$$Effort(random\ model) = \sum_{i=1}^{n} f_i \times p_i \Big/ \sum_{i=1}^{n} f_i$$

Before computing ER, we need to know the classification threshold for the prediction model *m*. In the literature, there are two popular methods to determine the classification threshold on a training set. The first method is called balanced-pf-pd (BPP) method. This method employs the ROC curve corresponding to *m* to determine the classification threshold. In the ROC curve, pd (probability of detection) is plotted against pf (probability of false alarm) [13]. Intuitively, the closer a point in the ROC curve is to the perfect classification point (0, 1), the better the model predicts. In this context, the "balance" metric

$$balance = 1 - \sqrt{(0-pf)^2 + (1-pd)^2} \Big/ \sqrt{2}$$

can be used to evaluate the degree of balance between pf and pd [13]. For a given training data set, BPP chooses the threshold having the maximum "balance". The second method is balanced-classification-error (BCE) method. Unlike BPP, BCE chooses the threshold to roughly equalize two classification error rates: false positive rate and false negative rate. As stated by Schein et al. [14], such an approach has the effect of giving more weight to errors from false positives in imbalanced data sets. This is especially important for vulnerable data, as they are typically imbalanced (i.e. most components have no vulnerabilities). For the simplicity of presentation, the effort reduction metrics under the BPP and BCE thresholds are respectively called "ER-BPP" and "ER-BCE". Our study will use "ER-BPP" and "ER-BCE" to evaluate the effort-aware classification effectiveness of a prediction model. However, it may be not adequate to use these two thresholds for model evaluation, as there are many other possible thresholds. In practice, it is possible that a model is good under the BPP and BCE thresholds but is poor under the other thresholds. Consequently, for software practitioners, it may be misleading to use the "ER-BPP" and "ER-BCE" metrics to select the best classification model from a number of alternatives. In this paper, we use an additional metric "ER-AVG" proposed by Zhou et al. [11] to alleviate this problem. For a given model, the "ER-AVG" metric is the average effort reduction of the model over all possible thresholds on the test data set. Therefore, the "ER-AVG" metric is indeed independent from specific thresholds. Consequently, it can provide a complete picture of the classification performance.

2) Effort-aware ranking performance evaluation

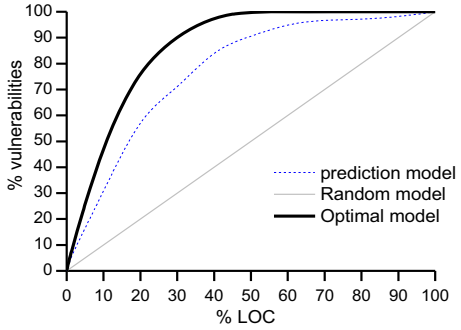


Fig. 1. LOC-based Alberg diagram

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

We use the cost-effectiveness measure CE proposed by Arisholm et al. [16] to evaluate the effort-aware ranking effectiveness of a vulnerability-proneness prediction model. The CE measure is based on the concept of the "SLOC-based" Alberg diagram. In this diagram, the x-axis is the cumulative percentage of SLOC of the components selected from the component ranking and the y-axis is the cumulative percentage of vulnerabilities found in the selected components. Consequently, each vulnerability-proneness prediction model corresponds to a curve in the diagram. Fig. 1 is an example "SLOC-based" Alberg diagram showing the ranking performance of a prediction model *m* (in our context, *m* could be the T model, the M model, the T' model, and the M' model). To compute CE, we also consider two additional curves, which respectively correspond to "random" model and "optimal" model. In the "random" model, components are randomly selected to test or inspect. In the "optimal" model, components are sorted in decreasing order according to their actual vulnerability densities.

Based on this diagram, the effort-aware ranking effectiveness of the prediction model m is defined as follows [16]:

$$CE_{\pi}(\mathrm{m}) = \frac{Area_{\pi}(\mathrm{m}) - Area_{\pi}(\text{random model})}{Area_{\pi}(\text{optimal model}) - Area_{\pi}(\text{random model})}$$

Here, $Area_{\pi}(m)$ is the area under the curve corresponding to model *m* for a given top $\pi \times 100\%$ percentage of SLOC. The cut-off value π varies between 0 and 1, depending on the amount of available resource for testing or inspecting components. Indeed, $CE_{\pi}(m)$ is the normalized "average recall" of vulnerabilities in relation to the optimal model and the random model over the interval from 0%SLOC to $\pi \times$ 100%SLOC. The range of $CE_{\pi}(m)$ is [-1, 1] and a larger value means a better ranking effectiveness. In the literature, $CE_{\pi}(m)$ has been considered as an effective way to measure the ranking performance of fault-proneness prediction models [17].

F. Data analysis method

In the following, we describe the data analysis method for investigating the three research questions. To obtain an adequate and realistic assessment, we examine RQ1, RQ2, and RQ3 with respect to both ranking and classification scenarios under 10 times 10-fold cross-validation and across-project prediction.

In order to answer RQ1, we need to compare the prediction effectiveness of text mining based models with metrics based models under the cross-validation and across-project prediction in ranking scenario. As mentioned before, the prediction effectiveness is measured by CE in the ranking scenario. We choose the measure of $CE_{0.1}$ and $CE_{0.2}$ in our study. If the measures of text mining based models are better than those of software metrics models, we can conclude that text mining based models outperform metrics based models in the context of effort-aware software vulnerability prediction in the ranking scenario.

In order to answer RQ2, we need to compare the prediction effectiveness of text mining based models with metrics based models under the cross-validation and across-project prediction in classification scenario. As mentioned before, the prediction effectiveness is measured by ER-BCE, ER-BPP, and ER-AVG in the classification scenario. If the measures of text mining based models are better than those of software metrics models, we can conclude that text mining based models outperform metrics based models in the context of effort-aware software vulnerability prediction in the classification scenario.

In order to answer RQ3, we need to investigate what the difference between the prediction effectiveness of the two types of models. We investigate whether they have a significant difference on the prediction effectiveness by the Wilcoxon signed-rank test. In particular, we examine whether a difference is significant at the significance level of 0.05 using the Benjamini-Hochberg (BH) corrected p-values [18]. If the statistical test shows that a significant difference exists, we further examine whether the magnitude of the difference is important from the viewpoint of practical application using the Cliff's δ [19], which is used for median comparison. The magnitude of the difference is considered either trivial ($|\delta| <$ 0.147), small (0.147-0.33), moderate (0.33-0.474), or large (> 0.474) [20]. These analyses will help determine how well text mining based models perform in predicting vulnerable components compared with metrics based models. If the different is trivial or small, we conclude that text mining based models are not more cost-effective when taking into account the cost to build and apply the models in the context of effort-aware vulnerability prediction.

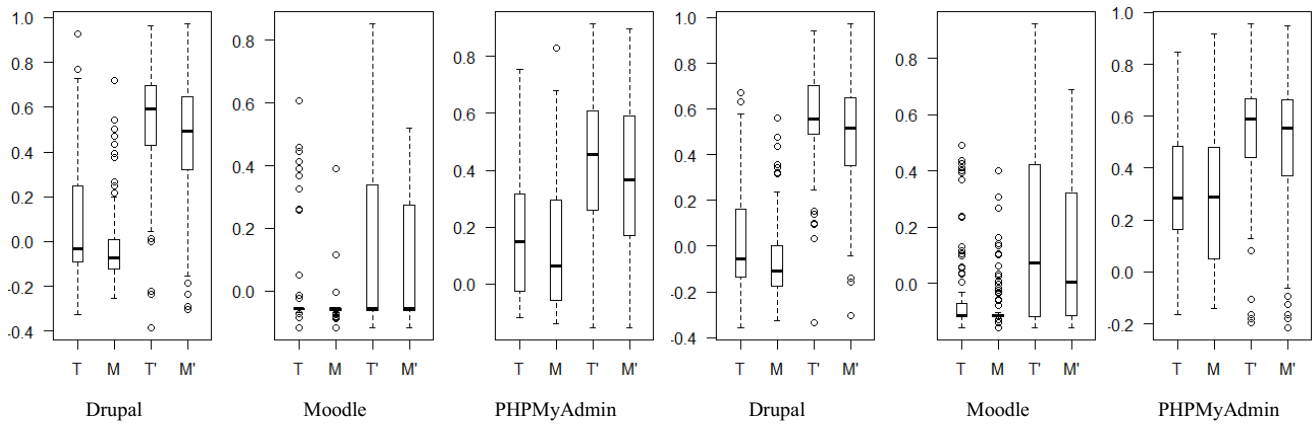
IV. Experimental Results

In this section, we report the experimental results in detail. In section IV-A, we report the results from comparing the effort-aware ranking performance for the text mining based and the metrics based models (RQ1). In section IV-B, we report the results from comparing the effort-aware classification performance for these two kinds of models (RQ2). In section IV-C, we report the results from statistically significant test and effect sizes of the differences between text mining based and metrics based models to examine whether text mining based model is more cost-effective from the viewpoint of practical application (RQ3).

A. RQ1: In the ranking scenario, do text mining based models outperform software metrics based models in the context of effort-aware vulnerability prediction?

In order to answer RQ1, we first use the procedure described in section III to build the T model (i.e. the text mining based model), the M model (i.e. the metrics based model), the T' model (i.e. the improved T model), and the M' model (i.e. the improved M model) on each data set. Then, we compare the ranking effectiveness of the following two pairs of models under cross-validation prediction and across-project prediction: 1) the T model vs the M model and 2) the T' model vs the M' model.

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.



(a) $CE_{0.1}$ (b) $CE_{0.2}$

Fig. 2. Ranking performance comparison for the T vs M and T' vs M' under 10 times 10-fold cross-validation

Fig. 2(a) and Fig. 2(b) respectively employ the box-plot to describe the distributions of the CE0.1 and CE0.2 obtained from 10 times 10-fold cross-validation for the T model, the M model, the T' model, and the M' model. For each model, the box-plot shows the median (the horizontal line within the box), the 25th and 75th percentiles (the lower and upper sides of the box). In practice, practitioners are more interested in the ranking performance of a prediction model at the top fraction. Therefore, we report the CE performances at $\pi = 0.1$ and 0.2 for each model.

From Fig. 2, we have the following observations:

- *Performance comparison for T vs M.* According to CE0.1, for the Drupal and the Moodle applications, the T model has a slightly larger median than the M model. However, the median $CE_{0.1}$ are lower than 0, indicating that both the T model and the M model perform worse than the random model in ranking top "10%-effort" files (i.e. the top ranked files that made up 10% of the entire effort required to inspect all files). For the PHPMyAdmin application, the T model has a larger median $CE_{0.1}$ than the M model. According to $CE_{0.2}$, we have similar observations. The core observation is that, from the viewpoint of effort-aware ranking scenario, the T model performs a little better than the M model but worse than the random model in most cases under 10 times 10-fold cross-validation.
- *Performance comparison for T' vs M'.* According to $CE_{0.1}$, the T' model has a larger or similar median than the M' models for all applications. Besides, the median $CE_{0.1}$, except the at Moodle application, are all around 0.5 which substantially outperform the random model, the T model, and the M model. This indicates that using the Pr/SLOC can improve the effort-aware ranking performance. According to $CE_{0.2}$, the T' model has a larger median than the M' model. In particular, both T' and M' outperform the random model as well as the T model and the M model. The core observation is that, from the viewpoint of effort-aware ranking scenario, the T' model outperforms the M' model and furthermore using the predicted vulnerable-proneness per *SLOC* can substantially improve the ranking performance.

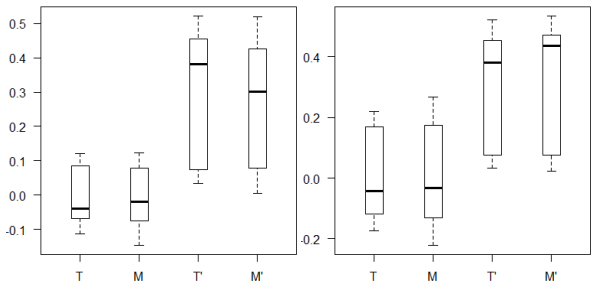


Fig. 3. Ranking performance comparison for T vs M and T' vs and M' under across-project prediction

Overall, the above observations suggest that text mining based models (i.e. the T model and the T' model) outperform software metrics based models (i.e. the M model and the M' model) in effort-aware vulnerability ranking under cross-validation prediction. Furthermore, using the predicted vulnerable-proneness per *SLOC* can substantially improve the ranking performance.

Fig. 3(a) and Fig. 3(b) respectively employ the box-plot to describe the distributions of the $CE_{0.1}$ and $CE_{0.2}$ obtained from across-project prediction for the T model, the M model, the T' model, and the M' model.

From Fig. 3, we have the following observations:

- *Performance comparison for T vs M.* According to $CE_{0.1}$, the T model has a similar median with the M model. The *median* $CE_{0.1}$ is lower than 0, indicating that both of these two models perform worse than the random model in ranking top "10%-effort" files. According to $CE_{0.2}$, we have similar observations. The core observation is that, from the viewpoint of effort-aware ranking scenario, the T model performs similarly to the M model but worse than the random model under across-project prediction.

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

- *Performance comparison for T' vs M'*. According to $CE_{0.1}$, the T' model has a larger median than the M' model. According to $CE_{0.2}$, the T' model has a smaller median than the M' model. Those median $CE_{0.1}$ and median $CE_{0.2}$ are all around 0.5 which substantially outperform the random model, the T model, and the M model. This also indicates that using the Pr/SLOC can improve the effort-aware ranking performance. The core observation is that, from the viewpoint of effort-aware ranking scenario, the T' model outperforms the M' model and furthermore using the predicted vulnerable-proneness per *SLOC* can substantially improve the ranking performance.

Overall, the above observations suggest that text mining based models do not always outperform software metrics based models in effort-aware vulnerability ranking under across-project prediction. Furthermore, we also find that using the predicted vulnerable-proneness per SLOC can substantially improve the ranking performance.

Combing the results from Fig. 2 and Fig. 3, in the effort-aware vulnerability ranking, we do not have enough evidence to support that text mining based models are superior to software metrics based models in both prediction settings. However, we found that the T' model and the M' model respectively substantially outperform the T model and the M model. Therefore, we should use the predicted vulnerable-proneness per SLOC for vulnerabilities ranking.

B. RQ2: In the classification scenario, do text mining based models outperform software metrics based models in the context of effort-aware vulnerability prediction?

In order to answer RQ2, we compare the prediction effectiveness of the following two pairs of models with respect to classification under cross-validation prediction and across-project prediction: 1) the T model vs the M model, and 2) the T' model vs the M' model.

Fig. 4(a), Fig. 4(b), and Fig. 4(c) respectively employ the box-plot to describe the distributions of the ER-BPP, ER-BCE, and ER-AVG obtained from 10 times 10-fold cross-validation for the T model, the M model, the T' model, and the M' model.

From Fig. 4, we have the following observations:

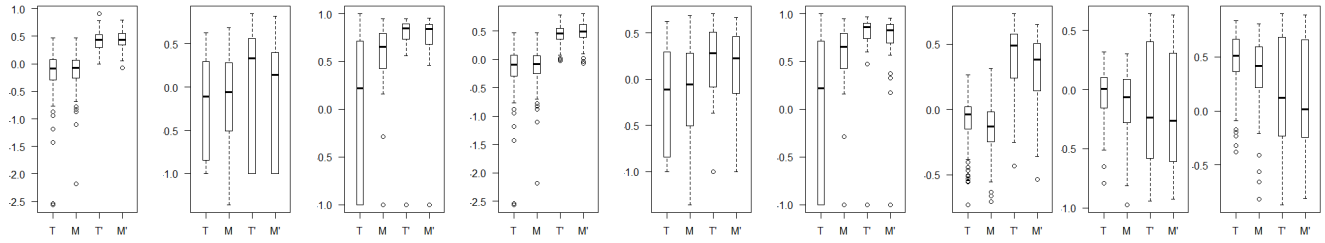
- *Performance comparison for T vs M*. According to ER-BPP, except for the PHPMyAdmin, the T model has a similar median with the M model. According to ER-BCE, We have similar observations. According to ER-AVG, the T *model* has a larger median than the M model. Most of the medians are less than 0, indicating that both the T and M models perform worse than the random model in effort-aware vulnerability classification. The core observation is that, from the viewpoint of effort-aware vulnerability classification, there is no consistent order between the T model and the M model.
- *Performance comparison for T' vs M'*. According to ER-BPP, the T' model has a similar or larger median than the M' model. According to ER-BCE and ER-AVG, we have similar observations. Besides, most of the median ER-BPP/ER-BCE/ER-AVG are around or larger than 0.5, indicating that they substantially outperform the random model. Furthermore, the T' model and the M' model have a larger median than the T model and the M model. In other words, using the Pr/SLOC as the predictive risk can improve the effort-aware classification performance. The core observation is that, from the viewpoint of effort-aware vulnerability classification, the T' model performs similarly to the M' model.

Overall, the above observations suggest that, in most cases, the text mining based models (i.e. the T and T' models) performs similarly to software metrics based models (i.e. the M and M' models) in the context of effort-aware vulnerability classification under cross-validation prediction. Furthermore, in most cases, using the predicted vulnerable-proneness per *SLOC* as the predictive vulnerability can substantially improve the classification performance.

Fig. 5(a), Fig. 5(b), and Fig. 5(c) respectively employ the box-plot to describe the distributions of the ER-BPP, ER-BCE, and ER-AVG obtained from across-project prediction for the T model, the M model, the T' model, and the M' model.

From Fig. 5, we have the following observations:

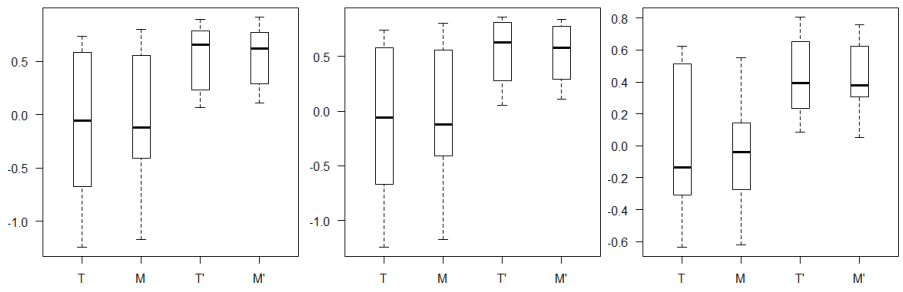
- *Performance comparison for T vs M*. According to ER-BPP, the T model has a similar median with the M model. According to ER-BCE and ER-AVG, we have similar observations. The median ER-BPP/ER-BCE/ER-AVG are less than 0, indicating that both of these two kinds of models perform worse than the



(a) ER-BPP (b) ER-BCE (c) ER-AVG

Fig. 4. Classification performance comparison for the T, M, T', and M' models under 10 times 10-fold cross-validation

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.



ER-BPP ER-BCE ER-AVG

Fig. 5. Classification performance comparison for the T vs M and T' vs M' models under across-project prediction

random model in the context of effort-aware vulnerability classification. The core observation is that, from the viewpoint of effort-aware classification scenario, the T model performs similarly to the M model but worse than the random model under across-project prediction.

- *Performance comparison for T' vs M'*. According to ER-BPP, the T' model has a similar median with the M' model. According to ER-BCE and ER-AVG, we have similar observations. Those medians are all around 0.5, substantially outperforming the random model, the T model, and the M model. This indicates that using the Pr/SLOC can improve the effort-aware classification performance. The core observation is that, from the viewpoint of effort-aware classification scenario, the T' model perform similarly to the M' model and using the predicted vulnerable-proneness per *SLOC* can substantially improve the classification performance.

Overall, the above observations suggest that text mining based models perform similarly to software metrics based model in effort-aware vulnerability classification under across-project prediction. Furthermore, we find that using the predicted vulnerable-proneness per SLOC can substantially improve the classification performance.

Combing the results from Fig. 4 and Fig. 5, in the effort-aware vulnerability classification, we find that the text mining based models have a similar classification performance with software metrics based models under both the cross-validation and across-project prediction settings. Besides, we found that the T' model and the M' model respectively substantially outperform the T model and the M model. Therefore, we should use the predicted vulnerable-proneness per SLOC for vulnerability classification.

C. *RQ3: Are text mining based models more cost-effective than software metrics based models in effort-aware vulnerability prediction?*

In section IV-A and section IV-B, we find that the T' and M' models which using the predicted vulnerable-proneness per *SLOC* as the predictive vulnerability perform better. Therefore, in order to answer RQ3, we first use Wilcoxon signed-rank test to test whether the differences between the T' model and the M' model is statistically significant. Then, we use the Cliff's δ to examine whether the magnitude of the difference between the T' model and the M' model are important from the viewpoint of practical application under both cross-validation and across-project prediction.

Table III and Table IV respectively presents the BH corrected p-value by the Wilcoxon signed-rank test and the effect size by the Cliff's δ for the comparison of the T' model and the M' model under cross-validation. In Table III, the first column is the application name. The second to the sixth columns respectively show the p-value from Wilcoxon signed-rank test with respect to $CE_{0.1}$, $CE_{0.2}$, ER-BPP, ER-BCE, and ER-AVG. In Table IV, the first column is the application name. The second to the sixth columns respectively show the Cliff's δ. The positive sign indicates that the T' model outperforms the M' model. In contrast, the negative sign indicates that the T' model is worse than the M' model. From Table III and Table IV, we have the following observations. According to the $CE_{0.1}$ and $CE_{0.2}$, except the $CE_{0.2}$ at PHPMyAdmin, the others are significant at the significant level of 0.05. According to ER-BPP/ER-BCE/ER-AVG, except ER-BPP at Drupal and PHPMyAdmin, ER-AVG at Moodle and PHPMyAdmin, the others are significant at the significant level of 0.05. Overall, we do not have enough evidence to support that the difference between the T' and M' models is significant in both ranking and classification prediction. According to Cliff's δ, most of the effect sizes are trivial.

TABLE III. THE P-VALUES FROM WILCOXON SIGNED-RANK TEST FOR THE COMPRISON OF T' AND M' UNDER 10 TIMES 10-FOLD CROSS-VALIDATION

Application	*CE0.1*	*CE0.2*	*ER-BPP*	*ER-BCE*	*ER-AVG*
Drupal	<0.001	<0.001	0.103	0.012	<0.001
Moodle	0.014	0.022	0.045	0.012	0.651
PHPMyAdmin	0.014	0.165	0.096	0.002	0.137

TABLE IV. THE EFFECT SIZES FROM CLIFF'S δ FOR THE COMPARISON OF T' AND M' UNDER 10 TIMES 10-FOLD CROSS-VALIDATION

Application	*CE0.1*	*CE0.2*	*ER-BPP*	*ER-BCE*	*ER-AVG*
Drupal	+0.233	+0.165	-0.076	-0.141	+0.245
Moodle	+0.061	+0.040	+0.136	+0.088	+0.056
PHPMyAdmin	+0.109	+0.074	+0.106	+0.159	+0.032

TABLE V. THE P-VALUES FROM WILCOXON SIGNED-RANK TEST FOR THE COMPRISON OF T' AND M' UNDER ACROSS-PROJECT PREDICTION

CE0.1	*CE0.2*	*ER-BPP*	*ER-BCE*	*ER-AVG*
< 0.001	< 0.001	0.721	0.003	0.558

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

TABLE VI. THE EFFECT SIZES FROM CLIFF'S δ FOR THE COMPARISON OF T' AND M' UNDER CCROSS-PROJECT PREDICTION

CE0.1	*CE0.2*	*ER-BPP*	*ER-BCE*	*ER-AVG*
+0.134	+0.326	+0.053	+0.113	-0.066

Table V and Table VI respectively present the BH corrected p-value by the Wilcoxon signed-rank test and the effect size by the Cliff's δ for the comparison of the T' and M' models under across-project prediction. From Table V and Table VI, we have the following observations. From Table V, according to the $CE_{0.1}$ and $CE_{0.2}$, the differences between T' and M' are significant at the significant level of 0.05. According to ER-BPP/ER-BCE/ER-AVG, only ER-BCE is significant. Overall, we do not have enough evidence to support that the differences between T' and M' is significant in both ranking and classification prediction. According to Cliff's δ, most of the effect sizes are trivial.

The core observation, from the viewpoint of effort-aware evaluation, we do not have enough evidence to conclude that the difference between the T' model and the M' model is significant. Furthermore, most of the effect sizes are trivial according to Cliff's δ. Therefore, from the viewpoint of practical application, software metrics based models have a predictive effectiveness comparable to text mining based models. When taking account the cost to build text mining based prediction models (such as extracting the term vectors from files and the time for building text mining prediction model), we suggest using software metrics based model for effort-aware vulnerability prediction.

D. Threats to Validity

External Threats. In our experiments, we use three web applications written by PHP as the subject systems. The experimental results drawn from these subject systems are consistent. Furthermore, the data sets from the three applications range from small to large to draw statistically meaningful conclusions. We believe that our study makes a strong complement research to effort-aware software vulnerability prediction. Nonetheless, we do not claim that our findings can be generalized to all systems written by any programming language, as the subject systems under study might not be representative of systems in general. To mitigate this threat, there is a need to use a wide variety of systems to replicate our study in the future.

Internal Threats. In our study, we treated only the lines of PHP code without HTML codes in each file as the proxy of code inspection effort. The HTML codes may have potential contributions to software vulnerability such as no measures to prevent illegal form input. However, this exclusion should not have a large influence on our conclusions, as the HTML codes occupy a small proportion.

V. CONCLUSION AND FUTURE WORK

In this paper, we perform a thorough empirical study to investigate the predictive power of text mining based models and software metrics based models in the context of effort-aware software vulnerability prediction. Our results show that text mining based models are only slightly better than or similar to software metrics based models in the context of effort-aware vulnerability prediction. Furthermore, in most cases, the effect sizes (i.e. the magnitude of the difference between these two kinds of models) are trivial. This indicates that, from the viewpoint of practical application, there is no important difference between these two kinds of models. However, text mining based models have higher modeling and applying cost than software metrics based models. Therefore, we suggest using software metrics based models rather than text mining based models in practice.

Our study only investigates the actual usefulness of effort-aware software vulnerability prediction for software systems written by the programming language of PHP. It is unclear whether they can be applied to software systems written by other programming languages with different features and vocabulary. In the future, an interesting work is hence to extend our current study to those software systems.

ACKNOWLEDGMENT

This work is supported by the National Key Basic Research and Development Program of China (2014CB340702), the National Natural Science Foundation of China (61432001, 91318301, 61272082, 61300051), and the Natural Science Foundation of Jiangsu Province (BK20130014).

REFERENCES

[1] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting Vulnerable Software Components via Text Mining," IEEE Trans. Softw. Eng., vol. 40, no. 10, pp. 993–1006, Oct. 2014.

[2] Walden, J. Stuckman, and R. Scandariato, "Predicting Vulnerable Components: Software Metrics vs Text Mining," in 2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE), 2014, pp. 23–33.

[3] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation," in Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings, 2004, pp. 149–155.

[4] D. Zhao and Y. Zhou, "Using unsupervised models to predict fault-proneness ranking for modules: an effort-aware experimental evaluation," Sci. Online China, 2013. http://www.paper.edu.cn/html/releasepaper/2013/04/22.

[5] D. Zhao, "Using clustering and voting to generate fault-prone module ranking," M.S. thesis, Nanjing University, China, 2013.

[6] A. G. Koru, D. Zhang, and H. Liu, "Modeling the Effect of Size on Defect Proneness for Open-Source Software," in 29th International Conference on Software Engineering - Companion, 2007. ICSE 2007 Companion, 2007, pp. 115–124.

[7] A. G. Koru, K. E. Emam, D. Zhang, H. Liu, and D. Mathew, "Theory of relative defect proneness," Empir. Softw. Eng., vol. 13, no. 5, pp. 473–498, Oct. 2008.

[8] A. G. Koru, D. Zhang, K. El Emam, and H. Liu, "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," IEEE Trans. Softw. Eng., vol. 35, no. 2, pp. 293–304, Mar. 2009.

[9] G. Koru, H. Liu, D. Zhang, and K. E. Emam, "Testing the theory of relative defect proneness for closed-source software," Empir. Softw. Eng., vol. 15, no. 6, pp. 577–598, Dec. 2010.

[10] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," Autom. Softw. Eng., vol. 17, no. 4, pp. 375–407, Dec. 2010.

[11] Y. Zhou, B. Xu, H. Leung, and L. Chen, "An In-depth Study of the Potentially Confounding Effect of Class Size in Fault Prediction," ACM Trans Softw Eng Methodol, vol. 23, no. 1, pp. 10:1–10:51, Feb. 2014.

[12] Y. Yang, Y. Zhou, H. Lu, L. Chen, Z. Chen, B. Xu, H. Leung, and Z. Zhang, "Are slice-based cohesion metrics actually useful in effort-aware

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.

post-release fault-proneness prediction? An empricial study," To apprear in IEEE Trans. Softw. Eng., 2015.

[13] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2–13, Jan. 2007.

[14] A. I. Schein, L. K. Saul, and L. H. Ungar, "A generalized linear model for principal component analysis of binary data," in Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, 2003, vol. 38, p. 46.

[15] L. Aversano, L. Cerulo, and C. Del Grosso, "Learning from Bug-introducing Changes to Prevent Fault Prone Code," New York, NY, USA, 2007, pp. 19–26.

[16] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," J. Syst. Softw., vol. 83, no. 1, pp. 2–17, Jan. 2010.

[17] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 1276–1304, Nov. 2012.

[18] Y. Benjamini and Y. Hochberg, "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing," J. R. Stat. Soc. Ser. B Methodol., vol. 57, no. 1, pp. 289–300, Jan. 1995.

[19] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," J. Syst. Softw., vol. 83, no. 1, pp. 2–17, Jan. 2010.

[20] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys," in annual meeting of the Florida Association of Institutional Research, 2006, pp. 1–33.

[21] B. Potter and G. McGraw, "Software security testing," IEEE Secur. Priv., vol. 2, no. 5, pp. 81–85, Sep. 2004.

[22] A. Arora and R. Telang, "Economics of software vulnerability disclosure," IEEE Secur. Priv., vol. 3, no. 1, pp. 20–25, Jan. 2005.

[23] Y. Shin and L. Williams, "Is Complexity Really the Enemy of Software Security?," in Proceedings of the 4th ACM Workshop on Quality of Protection, New York, NY, USA, 2008, pp. 47–50.

[24] Y. Shin and L. Williams, "An Empirical Model to Predict Security Vulnerabilities Using Code Complexity Metrics," in Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, New York, NY, USA, 2008, pp. 315–317.

[25] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," J. Syst. Archit., vol. 57, no. 3, pp. 294–313, Mar. 2011.

[26] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a Needle in a Haystack: Predicting Security Vulnerabilities for Windows Vista," in 2010 Third International Conference on Software Testing, Verification and Validation (ICST), 2010, pp. 421–428.

[27] H. Hata, O. Mizuno, and T. Kikuno, "Fault-prone module detection using large-scale text features based on spam filtering," Empir. Softw. Eng., vol. 15, no. 2, pp. 147–165, Sep. 2009.

[28] O O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, "Spam Filter Based Approach for Finding Fault-Prone Software Modules," in Proceedings of the Fourth International Workshop on Mining Software Repositories, Washington, DC, USA, 2007, p. 4–.

Authorized licensed use limited to: Nanjing University. Downloaded on February 23,2023 at 07:21:45 UTC from IEEE Xplore. Restrictions apply.