Improving Severity Prediction on Software Bug Reports using Quality Indicators

Cheng-Zen Yang, Kun-Yu Chen, Wei-Chen Kao, and Chih-Chuan Yang
Department of Computer Science and Engineering
Yuan Ze University
Chungli, Taoyuan 32003, Taiwan
{czyang,kychen12,wckao,tom}@syslab.cse.yzu.edu.tw

Abstract—**Recently, research has been conducted to explore the prediction schemes to identify the severity of bug reports. Several text mining approaches have been proposed to facilitate severity prediction. However, these studies mainly focus on the textual information of the bug reports. Other attributes of the bug reports have not been comprehensively discussed. In this paper, we investigate the influences of four quality indicators of bug reports in severity prediction. In an empirical study with the Eclipse dataset, the results show that considering these indicators can further improve the performance of a previous work employing only textual information.**

Keywords-bug reports; severity prediction; quality indicators; empirical study; performance evaluation

I. Introduction

To maintain software quality and correctness, software bug reports provide important assistance to software developers. Based on the information provided in the bug reports, the software developer find the locations for corrections and improve the software functionalities. Considering the efficiency of processing bug reports, a triager generally takes the responsibility for deciding the processing priority and dispatching them to the appropriate software developers. As discussed in [5,6,11], one critical factor influencing the priority decision is the bug severity which is an attribute to express the harshness resulted from the reported bug. Many bug tracking systems, such as BugZilla (http://www.bugzilla.org/), have defined their own severity levels. Reporters then follow these definitions to assign the severity levels to the bug reports.

Despite the clear distinctions among these severity levels, past studies show that a verification tool is needed to correct the reported severity attributes of some bug reports that have misassigned severity levels [5,6]. Therefore, Lamkanfi *et al.* use text mining algorithms to facilitate severity prediction [5,6]. In 2012, Yang *et al.* discussed the effectiveness of several well-known feature selection schemes to improve the performance of severity prediction [11]. Tian, Lo, and Sun proposed an information retrieval approach to improve the prediction performance by analyzing text information of bug reports [9]. Chaturvedi and Singh conducted an empirical study to investigate several machine learning techniques on severity prediction for open and closed source projects [3]. However, these studies focus on the machine learning techniques that only consider the textual information of the bug reports. Other attributes of the bug reports have not been comprehensively discussed in severity prediction.

In 2007, Bettenburg *et al.* conducted an empirical study on Eclipse to discuss the quality of the bug reports [1]. In this study, 16 report attributes were investigated with questionnaires to experienced developers. Among these attributes, three of them have the most important information for developers: *steps to reproduce*, *stack traces*, and *screenshots*. In 2008, Bettenburg *et al.* further discussed the structural information in bug reports [2]. They pointed out that the structural information, such as *patches*, *stack traces*, *source code*, and *enumerations*, may provide substantial hints for the problem causes. These characteristics of bug reports can be regarded as *quality indicators* from the aspect of developers. Since these quality indicators may reveal the grave situation for bug handling in addition to textual descriptions, the performance of severity prediction on bug reports can be enhanced by mining these indicators. Therefore, this paper aims to investigate the influences of these quality indicators on the severity prediction problem by an empirical study with a collected Eclipse dataset.

Based on the previous research work of Lamkanfi *et al.* [6], we consider four quality indicators in this study: *stack traces*, *report length*, *attachments*, and *steps to reproduce*. The experimental results on the Eclipse dataset show that considering these quality indicators effectively improves the prediction performance.

The rest of the paper is organized as follows. Section 2 provides a brief review on previous related work on severity prediction of software bug reports. In Section 3, we elaborate the prediction scheme. In Section 4, we describe the empirical experiments with the Eclipse dataset. Finally, Section 5 concludes the paper and describes our future work plan.

II. Related Work

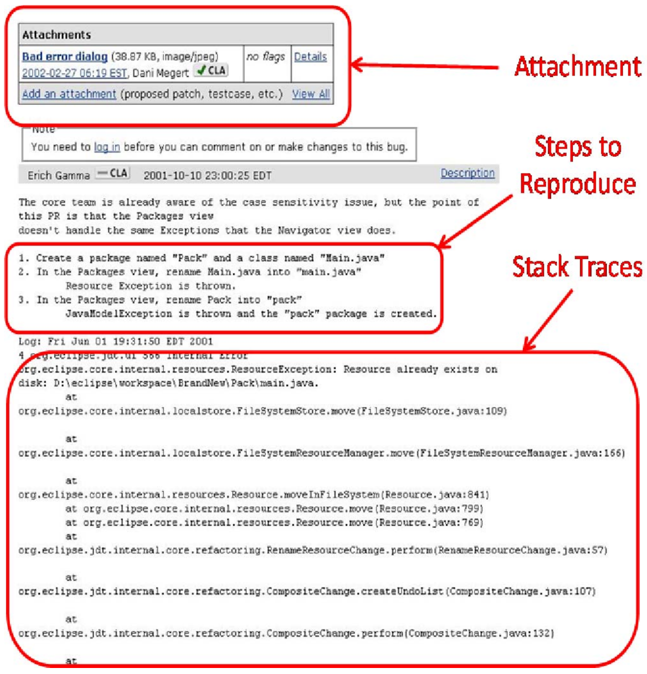
For the severity prediction problem, several research studies have been conducted recently [3,5,6,8,9,10,11]. In 2008, Menzies and Marcus proposed a severity prediction mechanism on the NASA dataset by analyzing the textual information with

978-1-4799-3279-5 /14/$31.00 ©2014 IEEE

rule-based text mining techniques [8]. Their approach handles fine-grained predictions based on the 5 severity levels defined by NASA. However, the performance of the proposed rule-based approach is dramatically unstable. In 2010, Lamkanfi *et al*. studied this problem using the Naïve Bayes classifiers to learn the severity semantics [5]. The bug reports are classified into two main coarse-grained groups: severe and non-severe. Their experimental results show that the severity semantics can be effectively learned from short summaries rather than long descriptions and some words are the potential severity indicators. Based on this work, the same research team further compared several classification schemes, such as Naïve Bayes, Naïve Bayes Multinomial, k-Nearest Neighbor, and Support Vector Machines, in 2011 [6]. Their experimental results show that NB Multinomial outperforms other schemes in the Area Under Curve (AUC) measures using the Receiver Operating Characteristic (ROC) evaluation.

To explore the effectiveness of different text mining techniques, Chaturvedi and Singh studied more classification schemes using the 5 severity levels of the NASA dataset in 2012 [3]. However, no one can outperform others for all software projects in their experiments. To demonstrate the impact of severity indicative words, Yang *et al*. studied several feature selection schemes to improve the performance of the NB Multinomial scheme [11]. Their experimental results show that the improvements can be achieved by adequately increasing the weights of severity indicators.

Based on a well-designed BM25-based similarity function, Tian, Lo, and Sun proposed a prediction scheme employing the duplication information in the bug reports [9]. To obtain the similarity between two bug reports, they considered four attributes: summary, description, product, and component. Then a nearest neighbor classification is employed to decide the severity level. The experimental results show that their scheme outperforms the work proposed by Menzies and Marcus. However, this scheme has many weighting and control parameters to be adjusted.



Attachments

Attachments		
Bad error dialog (38.87 KB, image/jpeg) 2002-02-27 06:19 EST, Dani Megert ✓CLA	no flags	Details
Add an attachment (proposed patch, testcase, etc.)		View All

Note
You need to log in before you can comment on or make changes to this bug.

Erich Gamma —CLA 2001-10-10 23:00:25 EDT Description

```
The core team is already aware of the case sensitivity issue, but the point of
this PR is that the Packages view
doesn't handle the same Exceptions that the Navigator view does.

1. Create a package named "Pack" and a class named "Main.java"
2. In the Packages view, rename Main.java into "main.java"
       Resource Exception is thrown.
3. In the Packages view, rename Pack into "pack"
       JavaModelException is thrown and the "pack" package is created.

Log: Fri Jun 01 19:31:50 EDT 2001
4 org.eclipse.jdt.ui 566 Internal Error
org.eclipse.core.internal.resources.ResourceException: Resource already exists on
disk: D:\eclipse\workspace\BrandNew\Pack\main.java.
       at
org.eclipse.core.internal.localstore.FileSystemStore.move(FileSystemStore.java:109)

       at
org.eclipse.core.internal.localstore.FileSystemResourceManager.move(FileSystemResourceManager.java:166)

       at
org.eclipse.core.internal.resources.Resource.moveInFileSystem(Resource.java:841)
       at org.eclipse.core.internal.resources.Resource.move(Resource.java:799)
       at org.eclipse.core.internal.resources.Resource.move(Resource.java:769)
       at
org.eclipse.jdt.internal.core.refactoring.RenameResourceChange.perform(RenameResourceChange.java:57)

       at
org.eclipse.jdt.internal.core.refactoring.CompositeChange.createUndoList(CompositeChange.java:107)

       at
org.eclipse.jdt.internal.core.refactoring.CompositeChange.perform(CompositeChange.java:132)

       at
```

Figure 1. Quality indicators in bug report #3759 of Eclipse.

Observing the important role of developers, Xuan et al. proposed a developer prioritization scheme to rank the contributions of the developers [10]. Then this scheme is used to improve the severity prediction of the work of Lamkanfi *et al.* in [5] by considering two priorities of reporters in software products and components. However, this scheme cannot get consistent improvements on all experimental software projects and components.

III. Severity Prediction

Like the work of Xuan *et al.* [10], we investigate the impacts of four quality indicators by considering these factors based on the previous 2011 work of Lamkanfi *et al.* [6]. The studied quality indicators are extracted from bug reports as shown in Fig. 1. This section first describes the prediction model. Then the prediction process employing the quality indicators is dictated.

A. Background of the Prediction Model

As discussed in the previous studies [5,6,11], the severity prediction problem in this paper is a binary classification problem in which a new bug report is classified into one of the following severity classes: *severe* or *non-severe*. For Eclipse, three severity levels are considered in the *severe* class: *Blocker*, *Critical*, and *Major*. Two other severity levels are considered in the *non-severe* class: *Minor* and *Trivial*. The bug reports with the *Normal* severity are ignored because many of them need further manual verification as indicated in [5]. The bug reports with the *Enhancement* severity are also ignored because they are actually the requests for function enhancements.

In this study, we also use the multinomial Naïve Bayes (MNB) classification model as the prediction scheme because the study in [6] shows that the MNB model outperforms the Naïve Bayes model, Support Vector Machines, and the 1-NN classification model. Therefore, the probability for a new bug report r_x to be classified into the severity s_i can be defined as $P(s_i\,|r_x)$ which is

$$P(s_i \mid r_x) = \frac{P(s_i)P(r_x \mid s_i)}{P(r_x)}.$$

Since $P(r_x)$ is the same for all s_i classes, it can be ignored. Thus, the probability $P(s_i|r_x)$ is proportional to $P(s_i)P(r_x|s_i)$. According to the Naïve Bayes assumption, each feature f_k in r_x contributes independently to the probability $P(r_x|s_i)$. Thus, if there are n features in r_x, this means that

$$\begin{aligned} P(s_i \mid r_x) &\propto P(s_i)P(r_x \mid s_i) \\ &= P(s_i)P(f_1, f_2, ..., f_n \mid s_i) \\ &= P(s_i)P(f_1 \mid s_i)P(f_2 \mid s_i) \times \cdots \times P(f_n \mid s_i) \\ &= P(s_i)\prod_{\forall k} P(f_k \mid s_i). \end{aligned}$$

The priori probability $P(s_i)$ is estimated as the number of the training bug reports in s_i divided by the number of the total bug reports in the training set. The probability $P(f_k|s_i)$ is estimated by

$$P(f_k \mid s_i) = \frac{N_k}{\sum_{\forall f_j \in F} N_j},$$

where N_k is the appearance number of feature f_k and F is the set of all features. However, the above formula for $P(f_k|s_i)$ has the zero-frequency problem [4,7]. Therefore, it is smoothed by

$$P(f_k \mid s_i) = \frac{1 + N_k}{|F| + \sum_{\forall f_j \in F} N_j}.$$

B. Prediction Process

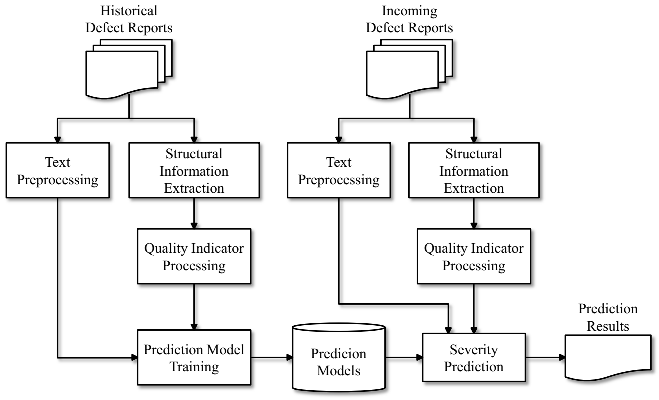
As shown in Fig. 2, the historical bug reports are used to train the prediction model using the aforementioned multinomial Naïve Bayes classifiers. All bug reports are first preprocessed to extract terms from the *summary* field as the textual information. As studied in [5,11], the *summary* field serves the best prediction basis because it usually contains terms, such as "fail", "crash", and "hanging", which have obvious severity tendencies. For textual information extraction, the following common text mining techniques are applied: tokenization, stopword removal, and stemming.

Then, the structural information in the *long description* field of bug reports is extracted to obtain the quality indicators. In this study, we extract two kinds of structural information from bug reports: *stack traces* (S) and *steps to reproduce* (R).

- Stack traces: the content of stack frames in the calling stack upon an exception.
- Steps to reproduce: the item list describing the sequence of actions to cause the bug

In addition to the S and R attributes, two more attributes are also extracted as the quality indicators: *attachments* (A) and *report length* (L).

- Attachments: the attached files to help bug fixation
- Report length: the content length of the long description providing debugging information



Historical Defect Reports
Incoming Defect Reports
Text Preprocessing
Structural Information Extraction
Text Preprocessing
Structural Information Extraction
Quality Indicator Processing
Quality Indicator Processing
Prediction Results
Prediction Model Training
Predicion Models
Severity Prediction

Figure 2. Processing flow to predict the severity of incoming bug reports.

IV. Experiments

To investigate the influences of quality indicators on severity prediction, we have conducted an empirical study on the Eclipse dataset as shown in Table I. The bug reports of two components were collected for experiments. In these two components, the bug reports whose status is "Invalid" or "New" were removed. The bug reports that have no long descriptions were also removed.

TABLE I. Eclipse bug reports used in the Experiments

Component	# of severe BR	# of non-severe BR	Period
JDT-UI	1544	1401	2001-2010
UI	3222	1231	2001-2010

The evaluation metrics are the Receiver Operating Characteristic (ROC) curves and their Area Under Curve (AUC) measures. The ROC curves illustrate the performance of different configurations by showing their true positive rates (TPR) against the false positive rates (FPR). TPR and FPR are defined as follows:

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN},$$

where *TP* is the number of correctly predicted severe reports, *FN* is the number of wrongly predicted non-severe reports, *FP* is the number of wrongly predicted severe reports, and *TN* is the number of correctly predicted non-severe reports.

In the experiments, we used a 10-fold cross validation approach to measure the performance of different configurations of four quality indicators. The aforementioned MNB classification model was used for all configurations. The baseline (T) was the configuration that only considered textual information of *summary* as [6]. For four quality indicators, their influences were first investigated by separately considering each of them with textual information: TS, TL, TA, and TR. Then we investigated the influences of all five kinds of information: TSLAR.

For these six configurations, we further studied two employment cases for these quality indicators: (a) using their Boolean values and (b) using their quantitative values. For the experiments considering the Boolean features, the binary values were decided according to their existence. Fig. 3 shows the ROC results of all configurations. Table II shows their AUC results. Note that TL is ignored because the Boolean value of the report length is always 1. From the experimental

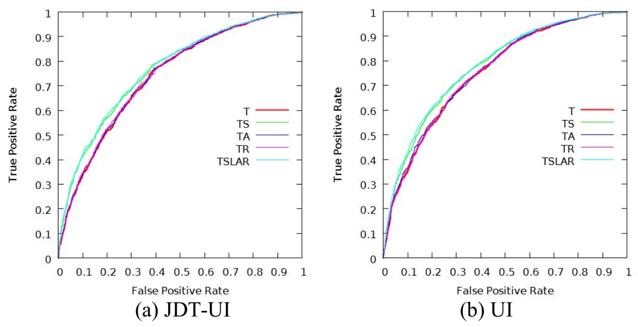


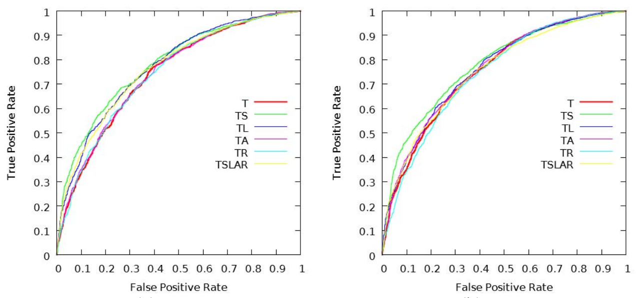
Figure 3. ROC results of 5 configurations considering their Boolean values.

TABLE II. AUC RESULTS OF 5 CONFIGURATIONS CONSIDERING THEIR BOOLEAN VALUES.

Config.	JDT-UI	UI
T (baseline)	0.740	0.759
TS	0.764	0.778
TA	0.742	0.758
TR	0.744	0.757
TSLAR	0.768	0.782

results, we can find that stack traces have primary contribution of performance improvement among all quality indicators. Furthermore, TSLAR outperforms other configurations.

For the experiments considering the quantities of quality indicators, the stack traces and the report length were measured in lines; the number of attachments was the number of files; the steps to reproduce were measured in the number of items. Fig. 4 shows the ROC results of all configurations. Table III shows their AUC results. From these experimental results, we can find that if the prediction model considers these quality indicators, all configurations can have performance



(a) JDT-UI (b) UI

Figure 4. ROC results of 6 configurations considering their quantitative values.

TABLE III. AUC RESULTS OF 6 CONFIGURATIONS CONSIDERING THEIR QUANTITATIVE VALUES.

Config.	JDT-UI	UI
T (baseline)	0.740	0.759
TS	0.779	0.778
TL	0.770	0.763
TA	0.743	0.761
TR	0.742	0.746
TSLAR	0.764	0.754

improvements for JDT-UI, and most configurations can have improvements for UI. Among all configurations, stack traces have the most effective contribution among for quality indicators. This is because most severe bug reports contain stack traces, but the non-severe bug reports seldom have this information in the Eclipse dataset.

V. CONCLUSION

In this paper, we investigate the influences of four quality indicators on severity prediction of bug reports: *stack traces*, *report length*, *attachments*, and *steps to reproduce*. In an empirical study on the collected Eclipse dataset, the results show that considering these quality indicators in a previous work can effectively improve the prediction performance in most cases.

According to current observations in this study, there are still many issues to be discussed. In our future work, additional quality indicators will be investigated to achieve more improvements. A comprehensive study will be also conducted for other software projects to validate the effectiveness of quality indicators.

ACKNOWLEDGMENT

This research was supported in part by the National Science Council of Taiwan under Grant NSC 101-2221-E-155-011-MY2.

REFERENCES

[1] N. Bettenburg, S. Just, A. Schröter, C. Wei R. Premraj, and T. Zimmermann, "Quality of Bug Reports in Eclipse", in *Proc. of the 2007 OOPSLA Workshop on Eclipse Technology eXchange (eclipse '07)*, pp. 21-25, 2007.

[2] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting Structural Information from Bug Reports," in *Proc. of the 5th International Working Conference on Mining Software Repositories (MSR '08)*, pp. 27-30, 2008.

[3] K. K. Chaturvedi, and V. B. Singh, "Determining Bug Severity using Machine Learning Techniques," in *Proc. of the 2012 CSI 6th International Conference on Software Engineering (CONSEG)*, 2012, pp. 1-6.

[4] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial Naive Bayes for Text Categorization Revisited," in *Proc. of the 17th Australian Joint Conference on Artificial Intelligence*, 2004, pp. 488-499.

[5] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the Severity of a Reported Bug," in *Proc. of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 1-10.

[6] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing Mining Algorithms for Predicting the Severity of a Reported Bug," in *Proc. of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, 2011, pp. 249-258.

[7] A. McCallum and K. Nigam, "A Comparison of Event Models for Naïve Bayes Text Classification," in *Proc. of the AAAI-98 Workshop on Learning for Text Categorization*, 1998, pp. 41-48.

[8] T. Menzies and A. Marcus, "Automated Severity Assessment of Software Defect Reports," in *Proc. of the 24th IEEE International Conference on Software Maintenance (ICSM 2008)*, 2008, pp. 346-355.

[9] Y. Tian, D. Lo, and C. Sun, "Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction," in *Proc. of the 2012 19th Working Conference on Reverse Engineering (WCRE '12)*, 2012, pp. 215-224.

[10] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer Prioritization in Bug Repositories," in *Proc. of the 2012 International Conference on Software Engineering (ICSE 2012)*, 2012, pp. 25-35.

[11] C.-Z. Yang, C.-C. Hou, W.-C. Kao, and I.-X. Chen, "An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection," in *Proc. of the 2012 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*, 2012, pp. 240-249.