Contemporary Engineering Sciences, Vol. 9, 2016, no. 19, 933 - 942
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2016.6695

Improving Predictions about Bug Severity by Utilizing Bugs Classified as Normal[1]

Kwanghue Jin, Amarmend Dashbalbar, Geunseok Yang and Byungjeong Lee*

Dept. of Computer Science and Engineering, University of Seoul
Siribdae-ro 163, Dongdaemun-Gu, Seoul 20504, Korea
*Corresponding author

Jung-Won Lee

Dept. of Electrical and Computer Engineering, Ajou University
16-1 Woncheon, Suwon, Gyonggi-do, 443-749, Korea

Abstract

More bugs have been generated because today's software has become large and complex. Some of these bugs are critical, while others are trivial. Because accurate prediction of bug severity enables software developers to effectively solve software problems, that accuracy aids software development and project planning. Therefore, this study presents a reliable approach to improve predictions about bug severity by utilizing bugs classified as normal, which is the default level specified in a submitted report. This approach uses attributes as well as text information in bug reports to produce more accurate prediction results. Bug reports in open source projects such as Mozilla and Eclipse were used in the experiments. The result shows that this approach performs better than other studies.

Keywords: Bug Severity Prediction, Normal Bugs, Classification, Machine Learning, Software Development, Project Planning

---

[1] This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014M3C4A7030504).

1 Introduction

More bugs are generated now because today's software has become large and complex. Bug fixing is an important phase in large and complex software development [1, 2]. Some bugs are critical, while others are trivial. Thus, the order in which bugs are fixed should be appropriate to their different severity levels. Accurate prediction about a bug's severity informs software developers as to which bug needs to be fixed urgently [3], which is very helpful in software development and maintenance. Moreover, an efficient method for bug severity prediction would allow organizations to decrease the workload of their developers, reduce expenses, and efficiently plan a schedule. Text information from bug reports has been used in previous prediction studies. However, prediction methods about bug severity may not provide much reliability, because text information is only one part of a bug report and depends on the bug reporter. Furthermore, the methods did not use bugs classified as normal, which is the default level specified in submitted reports. Therefore, this study presents a reliable approach to improve prediction of bug severity by utilizing bugs deemed normal. The proposed approach also uses other attributes, as well as text information in bug reports, to produce more accurate results. Below are the contributions of this study.

- This study uses most of the bug reports except for those with an importance level of Enhancement. Other studies did not use the large portion of bug reports classified as normal by the reporter.

- This study uses other attributes, such as the bug reporter, the component, the product, and the severity, as well as text information, such as summary and description.

- This study provides comparisons with other studies and, by utilizing statistical verification, shows that this approach is better than others.

The rest of the paper is structured as follows. Section 2 and Section 3, respectively, describe some background and some related studies. Section 4 presents the severity prediction method, and Section 5 offers the experiment results. Finally, Section 6 concludes the paper.

Who	When	What	Removed	Added
michael.wenz	2013-01-02 07:23:25 EST	Status	NEW	ASSIGNED
		CC		michael.wenz
		Hardware	PC	All
		OS	Windows 7	All
michael.wenz	2013-01-04 09:59:55 EST	Status	ASSIGNED	RESOLVED
		Resolution	---	FIXED
		Assignee	gmp.graphiti-inbox	michael.wenz
		Whiteboard		Kepler M5 Theme_bugs
		Flags		kepler+
matthias.gorning	2013-01-25 03:39:20 EST	Status	RESOLVED	VERIFIED
		CC		matthias.gorning
michael.wenz	2013-01-25 06:02:21 EST	Whiteboard	Kepler M5 Theme_bugs	Kepler M5 Theme_bugs Juno SR2 RC2
		Flags		juno+

Fig. 1. Eclipse bug fixing history (Bug 397288)

Fig. 2. Eclipse bug report (Bug 397288)

2 Background

A bug report is a history that contains a variety of information on a bug or a study related to it. Usually, a bug report looks like just a history of deficiencies that were found at runtime. However, it is an effective factor for software development and maintenance, because it includes essential information for not only the bug-fixing task but for assessing software quality. A bug-fixing history is shown in Figure 1. The bug report for Eclipse bug number 397288 in Figure 2 shows the bug's status changed from NEW to ASSIGNED by Michael Wenz on 2013-01-02 at 07:23:25, and the status also changed from ASSIGNED to RESOLVED on 2013-01-04 at 09:59:55. Finally, it was VERIFIED by Matthias Gorning on 2013-01-25 at 03:39:20. A broad set of information is recorded through a bug's lifetime, including Status, Resolution, Alias, Assignee, Assignee's Real Name, Blocks, Bug ID, CC, Changed, Classification, Comment, Component, Content, Creation Date, Depends On…, Description, Hardware, Importance, Keywords, OS, Priority, Product, QA Contact, QA Contact's Real Name, See Also…, Severity, Summary, Tags, Target Milestone, URL, Version, Votes, Whiteboard, etc. It is possible to see all that information about Eclipse bug number 397288 from the bug's report, as shown in Figure 2. The bug's priority rank is P3, a severity level that is Normal. Its present status is VERIFIED, and the fixing status is FIXED. The other information mentioned above is all available in the bug report.

3 Related Work

In previous research on bug severity prediction, Lamkanfi et al. [4, 5] described text mining algorithms to find the most effective algorithm for predicting bug severity from its text information. Their study compared four text mining algorithms, including naïve Bayes (NB), multinomial naïve Bayes (MNB), support vector machine (SVM) and K-nearest neighbor [5]. Eclipse and GNOME bug databases were used as experiment datasets to examine the efficiency of these algorithms. The MNB classifier generally outperformed other classification. The study also proved that around 250 bug reports at each severity level is a suitable amount of data to train the classifier. Severity Issue Assessment (SEVERIS), an automated method that predicts the severity of a reported bug, was introduced in a study by Menzies et al. [6]. SEVERIS also combined well-known text mining and machine learning algorithms. The method was applied to datasets from five anonymous PITS (Project Information Tracking System) projects run by NASA's Independent Verification and Validation Facility. However, the dataset was not sufficient, since 1 to 617 bug reports were used per severity level, and five levels were defined in the paper. The experiment results for 79,000 terms in 775 bug reports ranged between 65% and 98% in terms of F-measure. This study showed that usage of text mining– and machine learning–methods makes it possible to automatically generate levels of bug severity from the free text entered into the PITS. Roy et al. [7] introduced an approach to improving prediction accuracy that

is based on bi-grams and text mining algorithms. The naïve Bayesian approach was adopted as the classifier, and the dataset used for experimentation was collected from large-scale open source projects such as Mozilla Firefox and Eclipse. The result of the study showed that terms like crash and deadlock have the highest probability for being included in reports of severe bugs. However, the approach could not perform decent classifications to identify non-severe bugs, because they have a broad set of terms that have very low frequency rates in the text information of the bug report. There are other studies [8, 9] that have the same background and experiment data as this research direction, which focuses on estimating the time that elapses when fixing a specific bug. Jin et al. [10] presented bug severity prediction by classifying normal bugs. However, the study did not describe how to predict bug severity in detail. Moreover, the evaluation result of the study may not be reliable, because it did not present statistical verification.

4 Bug Severity Prediction

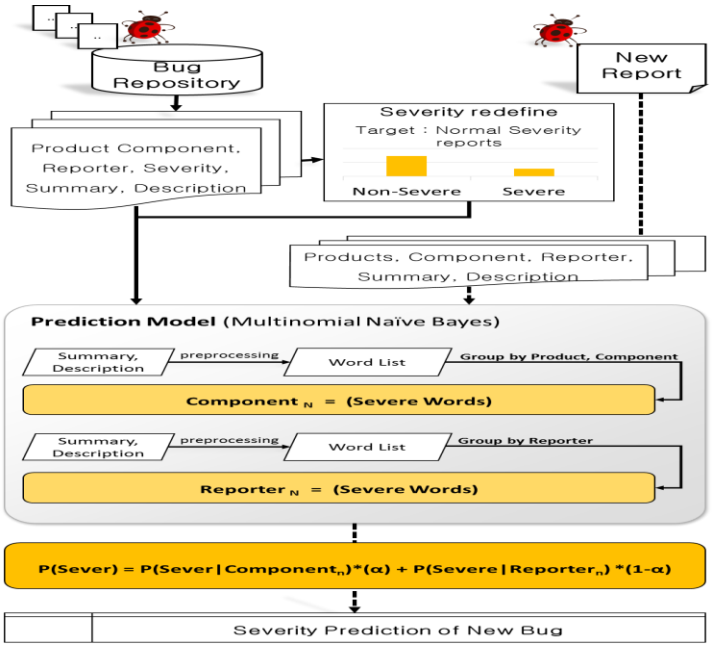

Fig. 3. Flow of proposed method

Information on bugs that are found during program runtime is an essential factor for the successful development and maintenance of a project. Therefore, predicting bug severity levels is fundamental for project planning; if bug severity levels are predicted inaccurately, it increases project expense and developers' workloads. This paper used text attributes, such as Summary and Description, and additional attributes (Product, Component, Reporter, and Severity) of the bug reports for the proposed classifier model, which was trained by a dataset that other studies did not use in order to make a reliable and highly accuracy classifier.

Tenfold cross validation [11] was used for the experiment on collected information from useful attributes of a bug report dataset. Figure 3 shows the process flow of the proposed method and how it predicts newly reported bugs by the redefined severity levels [10]. The process flow of the proposed method is defined as follows. First, the reported bugs are collected and features such as Product, Component, Reporter, Severity, Summary, and Description are extracted. Then, it proceeds to preprocessing tasks [12] for extracted text-attribute information like Summary and Description. Preprocessing tasks are: *tokenization*, which is a word-cutting method; *stemming*, which is a normalization method; and *preprocessing* for stop words, which eliminates unessential parts of words. Stemming cuts a word by predetermined rules and patterns, but not by linguistic analysis. For example, similar words like automates, automatic, and automation are all shortened to 'automat' instead. Stop-word removal cuts a word that is not very important to sentence meaning. If words like *he, she, it* and *is* are included in a sentence, the method removes those words in order to save memory, because they do not present any meaningful expression about bug severity. Natural Language Toolkit [13] ran the preprocess methods. After preprocessing the extracted text, the procedure reclassifies normal bugs in order to involve terms that are included in both normal and non-normal bug reports. For the comparison method, the preprocessed terms of the Summary and Description attributes are used in order to find similarities between terms pertaining to normal and other-than-normal severity levels. Most of the bugs where the initial attained severity level is normal were attained incorrectly. Thus, they need to be classified again, and reclassified bugs are used as the training dataset. The final stage for defining the bug severity prediction method is grouping all terms included in redefined bug reports by Reporter and Component fields into two classes: non-severe and severe. As shown in Table 1, trivial and minor levels are non-severe; major, critical and blocker levels are severe.

Table 1. Severity levels

Severity	**non-severe**	**severe**
Levels	trivial, minor	major, critical, blocker

By the prediction method defined above, when a reporter submits a new bug, it goes through preprocessing based on predefined information in the Component and Reporter fields before the model classifies the bug at a suitable level through the text similarity technique. The proposed model only predicts severe bug levels, such as major, critical, and blocker. Prediction of non-severe bugs is not 100% reliable because they may come about from typographical errors or text misalignment, or they may be functionally unimportant and easily fixed. For project planning, non-severe bug prediction is less useful than severe bug prediction. Predicting non-severe bugs is not feasible because, unlike severe bugs, generally, there is not enough detailed information on them in the reports.

The multinomial naïve Bayes approach helps to compare the similarity between two terms. Naïve Bayes classification is one of the simple classification techniques using supervised learning. Its classifier was made by training that depends on probability. The naïve Bayes classifier characteristically assumes that all property values are independent of each other. It shows the relationship between prior and posterior probabilities of two random variables based on Bayes' theorem, and works better than expected in complex real-life situations. MNB is a variant of NB, which is a polynomial algorithm for distributed classification.

5 Experiments

The experiment aimed to answer the following research question (RQ). Which method is able to predict the bug severity level more accurately? The proposed work is based on an MNB algorithm and uses normal data for training the method. Thus, in order to answer this question, the present study was compared with another study. To answer to the RQ, the null hypothesis is:

- H_0: This approach has no acceptable difference from the Lamkanfi et al. [4] approach.

The corresponding alternative hypothesis is:

- H_a: This approach shows better accuracy than the Lamkanfi et al. [4] study.

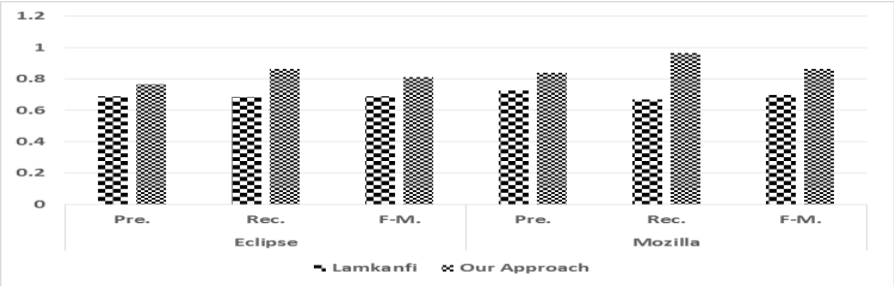

Fig. 4. Comparison result based on F-measure

The comparison of the proposed model with the Lamkanfi [4] method is shown in Figure 4 [10]. These methods were compared against the dataset used by Lamkanfi et al. Bug reports from 2001 to 2008 for Eclipse components, including the Eclipse user interface, Java Development Tool (JDT) user interface, and JDT and Mozilla bug reports from components such as Layout, Bookmarks, and Firefox General that were reported between 1997 and 2008, were used for the comparison dataset. The comparison result shows that the proposed method outperformed Lamkanfi et al.'s method, and that the F-measure of the Eclipse and Mozilla datasets has been improved.

Table 2. Result of statistical test

null hypothesis	**p-value**	**alternative**
H_0	p-value = 0.001546	H_a : Accept

The hypothesis above was tested in the R language using experiment results. First, the normality value was calculated for each hypothesis, and the normality value decides whether a Wilcox test or a t-test will be performed on the hypothesis. If the normality value is less than the threshold 0.05, a Wilcox test is performed; otherwise, the t-test is performed. Alternative hypothesis Ha was accepted because the result of the t-test was 0.001546, which means the proposed method is more accurate than the Lamkanfi et al. [4] approach. The results of the statistical test are shown in Table 2.

6 Conclusion

In this paper, text data and meta-field data in bug reports, which have not been used in other studies, were included in order to improve the prediction accuracy of the classification model. The training dataset also included bugs not used in other studies, which were designated as normal when the report was initially submitted. The bugs were classified using features extracted from text data and meta-fields of bug reports, which were used to predict their severity. The proposed method makes it possible to decrease the workload of developers and spare expenses that would otherwise be incurred in bug-fixing activities.

The experimental results show that the prediction accuracy is improved by the proposed model. However, fields aside from Component and Reporter were analyzed, but have not shown any effect on improvement. Analyzing the factors affecting the bug severity prediction through experiment is expected to further improve performance. Finally, for bugs submitted by reporters who are doing so for the first time, the accuracy may decline because only the Component field is used for prediction, since no past report history exists. Text information used for prediction is also objective. Thus, further research that presents objective criteria for prediction of bug severity is needed for the proposed method to be effective.

References

[1] K.K. Chaturvedi and V.B. Singh, Determining Bug Severity Using Machine Learning Techniques, *Proc. of CSI Sixth International Conference on Software Engineering,* (2012), 1-6.
http://dx.doi.org/10.1109/conseg.2012.6349519

[2] G.M. Tchamgoue, K.H. Kim and Y.K. Jun, Testing and Debugging Concurrency Bugs in Event-Driven Programs, *International Journal of Advanced Science and Technology*, **40** (2012), 55-68.

[3] E. Giger, M. Pinzger and H. Gall, Predicting the Fix Time of Bugs, *Proc. of International Workshop on Recommendation System for Software Engineering*, (2010), 52-56. http://dx.doi.org/10.1145/1808920.1808933

[4] A. Lamkanfi, S. Demeyer, E. Giger and B. Goethals, Predicting the Severity of a Reported Bug, *Proc. of IEEE Working Conference on Mining Software Repositories*, (2010), 1-10. http://dx.doi.org/10.1109/msr.2010.5463284

[5] A. Lamkanfi, S. Demeyer, Q. Soetens and T. Verdonck, Comparing Mining Algorithms for Predicting the Severity of a Reported Bug, *Proc. of European Conference on Software Maintenance and Reengineering*, (2011), 249-258. http://dx.doi.org/10.1109/csmr.2011.31

[6] T. Menzies and A. Marcus, Automated Severity Assessment of Software Defect Reports, *Proc. of International Conference Software Maintenance*, (2008), 346-355. http://dx.doi.org/10.1109/icsm.2008.4658083

[7] N.K.S. Roy and B. Rossi, Towards an Improvement of Bug Severity Classification, *Proc. of EUROMICRO Conference on Software Engineering and Advanced Applications*, (2014), 269-276. http://dx.doi.org/10.1109/seaa.2014.51

[8] C. Weiss, R. Premraj, T. Zimmermann and A. Zeller, How Long Will it Take to Fix This Bug?, *Fourth International Workshop on Mining Software Repositories*, (2007), 1. http://dx.doi.org/10.1109/msr.2007.13

[9] S. Kim and J.E. Whitehead, How Long Did it Take to Fix Bugs?, *Proc. of International Workshop on Mining Software Repositories*, (2006), 173-174. http://dx.doi.org/10.1145/1137983.1138027

[10] K. Jin, A. Dashbalbar, G. Yang, J.W. Lee, B. Lee, Bug Severity Prediction by Classifying Normal Bugs with Text and Meta-Field Information, *Advanced Science and Technology Letters*, **129** (2016), 19-24. http://dx.doi.org/10.14257/astl.2016.129.05

[11] R. Kohavi, A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, *Proc. of the 14th International Joint Conference on Artificial Intelligence*, (1995), 1137-1143.

[12] R. Feldman and S. James, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*, Cambridge University Press, 2006. http://dx.doi.org/10.1017/cbo9780511546914

[13] S. Bird, L. Edward and K. Ewan, *Natural Language Processing with Python*, O'Reilly Media Inc., 2009.

Received: May 1, 2016; Published: August 12, 2016