Automatic Bug Triage with Software

[1]Dhanashree Lohagaonkar, [2]Kanchan Kharat , [3]Omkar Bhise, [4]Abhijeet Korhale,
[5]Prof Mrs H.A.Shinde

[1,2,3,4] Student, Department of Computer Engineering All India Shri Shivaji Memorial Society Polytechinc, Kennedy road, Pune, Maharashtra, India.
[5]Lecturer, Department of Computer Engineering All India Shri Shivaji Memorial Society Polytechnic Kennedy Road, Pune, Maharashtra, India.

---***---

Abstract*—Software companies spend over 45 percent of cost in dealing with software bugs. An inevitable step of fixing bugs is bug triage, which aims to correctly assign a developer to a new bug. To decrease the time cost in manual work, text classification techniques are applied to conduct automatic bug triage. In this paper, we address the problem of data reduction for bug triage, i.e., how to reduce the scale and improve the quality of bug data. We combine instance selection with feature selection to simultaneously reduce data scale on the bug dimension and the word dimension. To determine the order of applying instance selection and feature selection, we extract attributes from historical bug data sets and build a predictive model for a new bug data set. We empirically investigate the performance of data reduction on totally 600,000 bug reports of two large open source projects, namely Eclipse and Mozilla. The results show that our data reduction can effectively reduce the data scale and improve the accuracy of bug triage. Our work provides an approach to leveraging techniques on data processing to form reduced and high-quality bug data in software development and maintenance.*

INTRODUCTION

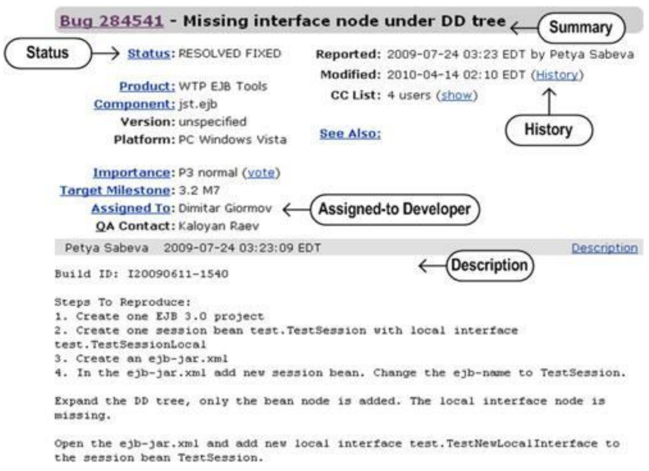
software repositories is an interdisciplinary domain, which aims to employ data mining to deal with software engineering problems In modern software development, software repositories are large-scale databases for storing the output of software development, e.g., source code, bugs, emails, and specifications. Tradi-tional software analysis is not completely suitable for the large-scale and complex data in software repositories [58]. Data mining has emerged as a promising means to handle software data (e.g., [7], [32]). By leveraging data mining techniques, mining software repositories can uncover inter-esting information in software repositories and solve real-world software problems.

A bug repository (a typical software repository, for storing details of bugs), plays an important role in managing soft-ware bugs. Software bugs are inevitable and fixing bugs is expensive in software development. Software companies spend over 45 percent of cost in fixing bugs [39]. Large soft-ware projects deploy bug repositories (also called bug track-ing systems) to support information collection and to assist.

BACKGROUND AND MOTIVATION

Background

Bug repositories are widely used for maintaining software bugs, e.g., a popular and open source bug repository, Bug-zilla [5]. Once a software bug is found, a reporter (typically a developer, a tester, or an end user) records this bug to the bug repository. A recorded bug is called a bug report, which has multiple items for detailing the information of repro-ducing the bug. In Fig. 1, we show a part of bug report for bug 284541 in Eclipse.[2] In a bug report, the summary and the description are two key items about the information of the bug, which are recorded in natural languages. As their names suggest, the summary denotes a general statement for identifying a bug while the description gives the details for reproducing the bug. Some other items are recorded in a bug report for facilitating the identification of the bug, such as the product, the platform, and the importance. Once a bug report is formed, a human triager assigns this bug to a developer, who will try to fix this bug. This developer is recorded in an item assigned-to. The assigned-to will change to another developer if the previously assigned developer cannot fix this bug. The process of assigning a correct developer for fixing the bug is called bug triage. For example, in Fig. 1, the developer Dimitar Giormov is the final assigned-to developer of bug 284541. A developer, who is assigned to a new bug report, starts to fix the bug based on the knowledge of historical bug fix-ing [36], [64]. Typically, the developer pays efforts to under-stand the new bug report and to examine historically fixed bugs as a reference (e.g., searching for similar bugs [54] and applying existing solutions to the new bug [28]).An item status of a bug report is changed according to the current result of handling this bug until the bug is completely fixed. Changes of a bug report are stored in an item history. Table 1 presents a part of history of bug 284541. This bug has been assigned to three developers and only the last developer can handle this bug correctly. Changing developers lasts for over seven months while fixing this bug only costs three days.



Bug 284541 - Missing interface node under DD tree

Status: RESOLVED FIXED
Reported: 2009-07-24 03:23 EDT by Petya Sabeva
Modified: 2010-04-14 02:10 EDT (History)
CC List: 4 users (show)
Product: WTP EJB Tools
Component: jst.ejb
Version: unspecified
Platform: PC Windows Vista
See Also:
Importance: P3 normal (vote)
Target Milestone: 3.2 M7
Assigned To: Dimitar Giormov
QA Contact: Kaloyan Raev

Petya Sabeva 2009-07-24 03:23:09 EDT Description

```
Build ID: I20090611-1540

Steps To Reproduce:
1. Create one EJB 3.0 project
2. Create one session bean test.TestSession with local interface
test.TestSessionLocal
3. Create an ejb-jar.xml
4. In the ejb-jar.xml add new session bean. Change the ejb-name to TestSession.

Expand the DD tree, only the bean node is added. The local interface node is
missing.

Open the ejb-jar.xml and add new local interface test.TestNewLocalInterface to
the session bean TestSession.
```

Motivation

Real-world data always include noise and redundancy [31]. Noisy data may mislead the data analysis techniques [66] while redundant data may increase the cost of data process-ing [19]. In bug repositories, all the bug reports are filled by developers in natural languages. The low-quality bugs accu-mulate in bug repositories with the growth in scale. Such

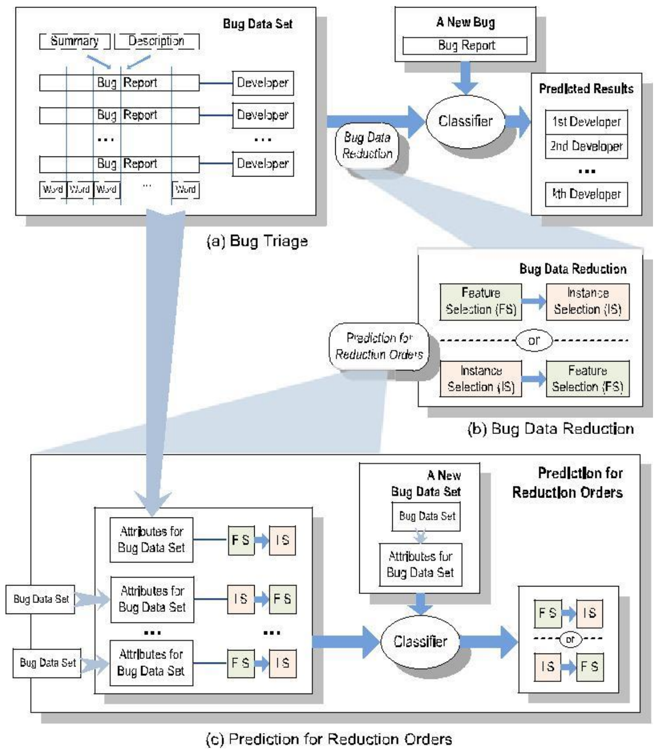


Fig. 2. Illustration of reducing bug data for bug triage. Sub-figure

(a) presents the framework of existing work on bug triage. Before train-ing a classifier with a bug data set, we add a phase of data reduction, in (b), which combines the techniques of instance selection and feature selection to reduce the scale of bug data. In bug data reduction, a prob-lem is how to determine the order of two reduction techniques. In (c), based on the attributes of historical bug data sets, we propose a binary classification method to predict reduction orders.

Example 1 (Bug 205900). Current version in Eclipse Europa discovery repository broken.

. . . [Plug-ins] all installed correctly and do not show any errors in Plug-in configuration view. Whenever I try to add a [diagram name] diagram, the wizard cannot be started due to a missing [class name] class . . .

In this bug report, some words, e.g., installed, show, started, and missing, are commonly used for describing bugs. For text classification, such common words are not helpful for the quality of prediction. Hence, we tend to remove these words to reduce the computation for bug tri-age. However, for the text classification, the redundant words in bugs cannot be removed directly. Thus, we want to adapt a relevant technique for bug triage.

To study the noisy bug report, we take the bug report of bug 201598 as Example 2 (Note that both the summary and the description are included).

Example 2 (Bug 201598). 3.3.1 about says 3.3.0.

Build id: M20070829-0800. 3.3.1 about says 3.3.0.

This bug report presents the error in the version dialog. But the details are not clear. Unless a developer is very familiar with the background of this bug, it is hard to find the details. According to the item history, this bug is fixed by the developer who has reported this bug. But the summary of this bug may make other developers confused. Moreover, from the perspective of data processing, espe-cially automatic processing, the words in this bug may be removed since these words are not helpful to identify this bug. Thus, it is necessary to remove the noisy bug reports and words for bug triage.

Algorithm 1. Data reduction based on FS ! IS

Input: training set T with n words and m bug reports, reduction order FS!IS

final number n_F of words, final number m_I of bug reports,

Output: reduced data set T_{FI} for bug triage

1) apply FS to n words of T and calculate objective values for all the words;

2) select the top n_F words of T and generate a training set T_F ;

3) apply IS to m_I bug reports of T_F ;

4) terminate IS when the number of bug reports is equal to or less than m_I and generate the final training set T_{FI} .

Algorithm 2. NaïveBayes Algorithm (Use to classifying data)-

Input- Training data set
Output-

1. To classify fixed bug from data set

2.To classify most solved bug on which type.

3.To assign bug to developer to most solved bugs as particular type of bug.

EXISTING SYSTEM:

A time-consuming step of handling software bugs is bug triage, which aims to assign a correct developer to fix a new bug. In traditional software development, new bugs are manually triaged by an expert developer, i.e., a human triage. Due to the large number of daily bugs and the lack of expertise of all the bugs, manual bug triage is expensive in time cost and low in accuracy. In manual bug triage in Eclipse, percent of bugs are assigned by mistake while the time cost between opening one bug and its first triaging is 19.3 days on average. To avoid the expensive cost of manual bug triage, existing work has proposed an automatic bug triage approach, which applies text classification techniques to predict developers for bug reports. In this approach, a bug report is mapped to a document and a related developer is mapped to the label of the document. Then, bug triage is converted into a problem of text classification and is automatically solved with mature text classification techniques, e.g., Naive Bayes. Based on the results of text classification, a human triage assigns new bugs by incorporating his/her expertise. However, large-scale and low-quality bug data in bug repositories block the techniques of automatic bug triage. .Since software bug data are a kind of free-form text data, it is necessary to generate well-processed bug data to facilitate the application.

we address the problem of data reduction for bug triage, i.e., how to reduce the bug data to save the labor cost of developers and improve the quality to facilitate the process of bug triage. Data reduction for bug triage aims to build a small-scale and high-quality set of bug data by removing bug reports and words, which are redundant or non-informative. In our work, we combine existing techniques of instance selection and feature selection to simultaneously reduce the bug dimension and the word dimension. The reduced bug data contain fewer bug reports and fewer words than the original bug data and provide similar information over the original bug data. We evaluate the reduced bug data according to two criteria: the scale of a data set and the accuracy of bug triage. To avoid the bias of a single algorithm, we empirically examine the results of four instance selection algorithms and four feature selection algorithm.

SOFTWARE REQUIREMENTS:

- Operating System : Windows 7
- Technology : Java and J2EE
- Web Technologies : Html, JavaScript, CSS
- IDE : Eclipse Juno
- Web Server : Tomcat
- Database : My SQL
- Java Version : J2SDK1.7

HARDWARE REQUIREMENTS:

- Hardware :Pentium Dual Core
- Speed :2.80 GHz
- RAM : 1GB
- Hard Disk : 20 GB
- Floppy Drive : 1.44 MB
- KeyBoard :Standard Windows Keyboard
- Mouse :Two or Three Button Mouse
- Monitor : SVGA

MODULE DESCRIPTION:

INSTANCE SELECTION:

Instance selection and feature selection are widely used techniques in data processing. For a given data set in a certain application, instance selection is to obtain a subset of relevant instances (i.e., bug reports in bug data) while feature selection aims to obtain a subset of relevant features (i.e., words in bug data). In our work, we employ the combination of instance selection and feature selection.

DATA REDUCTION:

In our work, to save the labor cost of developers, the data reduction for bug triage has two goals.

1) Reducing the data scale.

2) Improving the accuracy of bug triage.

DISADVANTAGES:

- We present the problem of data reduction for bug triage. This problem aims to augment the data set of bug triage in two aspects, namely

  a) To simultaneously reduce the scales of the bug dimension and the word dimension.

  b) To improve the accuracy of bug triage.

- We propose a combination approach to addressing the problem of data reduction. This can be viewed as an application of instance selection and feature selection in bug repositories.

PROPOSED SYSTEM:

In this part, we present the data preparation for applying the bug data reduction. We evaluate the bug data reduction on bug repositories of two large open source projects, namely Eclipse and Mozilla. Eclipse is a multi-language software development environment, including an Integrated Development Environment (IDE) and an extensible plug-in system; Mozilla is an Internet application suite, including some classic products, such as the Firefox browser and the Thunderbird email client. Up to December 31, 2011, 366,443 bug reports over 10 years.

RESULT:

We examine the results of bug data reduction on bug repositories of two projects, Eclipse and Mozilla. For each project, we evaluate results on five data sets and each data set is over 10,000 bug reports, which are fixed or duplicate bug reports. We check bug reports in the two projects and find out that 45.44 percent of bug reports in Eclipse and 28.23 percent of bug reports in Mozilla are fixed or duplicate.

CONCLUSION:

Bug triage is an expensive step of software maintenance in both labor cost and time cost. Our work provides a techniques on data processing to form reduced and high-quality bug data in software development and maintenance. The results of data reduction in bug triage to explore how to prepare a high quality bug data set and tackle a domain specific software task. To find out the potential relationship between the attributes of bug data sets and the reduction orders using predicting reduction orders.

REFERENCE

1. S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng., May 2010, pp. 481–490.

2. A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in Proc. 7th IEEE Working Conf. Mining Softw. Repositories, May 2010, pp. 1–10.

3. G. Lang, Q. Li, and L. Guo, "Discernibility matrix simplifica-tion with new attribute dependency functions for incomplete information systems," Knowl. Inform. Syst., vol. 37, no. 3, pp. 611–638, 2013.

4. D. Lo, J. Li, L. Wong, and S. C. Khoo, "Mining iterative generators and representative rules for software specification discovery," IEEE Trans. Knowl. Data Eng., vol. 23, no. 2, pp. 282–296, Feb. 2011.

5. Mozilla. (2014). [Online]. Available: http://mozilla.org/

6. D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in Proc. 6th Int. Working Conf. Mining Softw. Repositories, May 2009, pp.131-140.

7. G. Miao, L. E. Moser, X. Yan, S. Tao, Y. Chen, and N. Anerousis, "Generative models for ticket resolution in expert networks," in Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2010, pp. 733–742.

8. E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The design of bug fixes," in Proc. Int. Conf. Softw. Eng., 2013, pp. 332– 341.

9. J. A. Olvera-Lopez, J. A.Carrasco-Ochoa, J. F. Martınez-Trinidad, and J. Kittler, "A review of instance selection methods," Artif. Intell. Rev., vol. 34, no. 2, pp. 133–143, 2010.

10. J. A. Olvera-Lopez, J. F. Martınez-Trinidad, and J. A. Carrasco-Ochoa, "Restricted sequential floating search applied to object selection," in Proc. Int. Conf. Mach. Learn. Data Mining Pattern Rec-ognit., 2007, pp. 694–702.

11. R. S. Pressman, Software Engineering: A Practitioner's Approach, 7th
ed. New York, NY, USA: McGraw-Hill, 2010.

12. J. W. Park, M. W. Lee, J. Kim, S. W. Hwang, and S. Kim, "Costriage: A cost-aware triage algorithm for bug reporting sys-tems," in Proc. 25th Conf. Artif. Intell., Aug. 2011, pp. 139–144.

13. J. C. Riquelme, J. S. Aguilar-Ruız, and M. Toro, "Finding represen-tative patterns with ordered projections," Pattern Recognit., vol. 36pp.1009–1018, 2003.