NextBug: A Tool for Recommending Similar Bugs in Open-Source Systems

Henrique S. C. Rocha[1], **Guilherme A. de Oliveira**[2], **Humberto T. Marques-Neto**[2], **Marco Túlio O. Valente**[1]

[1] Department of Computer Science
Federal University of Minas Gerais (UFMG)
Belo Horizonte – MG – 31.270-901 – Brazil

[2]Department of Computer Science
Pontifical Catholic University of Minas Gerais (PUC Minas)
Belo Horizonte – MG – 30.535-901 – Brazil

henrique.rocha@dcc.ufmg.br, guilherme.oliveira.753469@sga.pucminas.br

humberto@pucminas.br, mtov@dcc.ufmg.br

Abstract. *Due to the characteristics of the maintenance process of open-source systems, grouping similar bugs to improve developers productivity is a challenging task. In this paper, we proposed and evaluate a tool, called NextBug, for recommending similar bugs in open-source systems. NextBug is implemented as Bugzilla plug-in and it was design to help maintainers selecting the next bug he/she would fix. We also report an experience on using NextBug with 109,145 bugs previously reported for Mozilla products.*

Video URL: <*http://youtu.be/Tt69zVobnF8*>

1. Introduction

Considering the great importance, the costs, and the increasing complexity of software maintenance activities, most organizations usually maintain their systems by performing tasks periodically, i.e., maintenance requests are grouped and implemented as part of large software projects [Tan and Mookerjee 2005; Aziz et al. 2009; Junio et al. 2011; Marques-Neto et al. 2013]. On the other hand, open-source projects typically adopt continuous maintenance policies where the maintenance requests are addressed by maintainers with different skills and commitment levels, as soon as possible, after being registered in an issue tracking platform, such as Bugzilla and Jira [Mockus et al. 2002; Tan and Mookerjee 2005; Liu et al. 2012].

However, this process is usually uncoordinated, which results in a high number of issues from which many are invalid or duplicated [Liu et al. 2012]. In 2005, a certified maintainer from the Mozilla Software foundation made the following comment on this situation: "everyday, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" [Anvik et al. 2006]. The dataset formed by bugs reported for the Mozilla projects indicates that, in 2011, the number of reported issues per year increased approximately 75% when compared to 2005. In this context, tools to assist in the issue processing would be very helpful and could contribute to increase the productivity of open-source systems development.

In this paper, we claim that a very simple form of periodic maintenance policy can be promoted in open-source systems by recommending similar maintenance requests to maintainers whenever they manifest interest in handling a given request. Suppose that a developer has manifested interest in a bug with a textual description d_i. In this case, we rely on text mining techniques to retrieve open bugs with descriptions d_j similar to d_i and we recommend such bugs to the maintainers.

More specifically, we present *NextBug*, a tool to recommend similar bugs to maintainers based on the textual description of each bug stored in Bugzilla, an issue tracking system widely used by open-source projects. The proposed tool is compatible with the software development process followed by open-source systems for the following reasons: (a) it is based on recommendations and, therefore, maintainers are not required to accept extra bugs to fix; (b) it is a fully automatic and unsupervised approach which does not depend on human intervention; and (c) it relies on information readily available in Bugzilla. Assuming the recommendations effectively denote similar bugs and supposing that the maintainers would accept the recommendations pointed out by NextBug, the tool can contribute to introduce gains of scale similar to the ones achieved with periodic policies [Banker and Slaughter 1997]. We also report a field study when we populated NextBug with a dataset of bugs reported for Mozilla systems.

The remainder of this paper is organized as follows. Section 2 discuss tools for finding duplicated issue reports in bug tracking systems and also tools that assign bugs to developers. The architecture and the central features of NextBug are described in Section 3. An example of usage is presented in Section 4. Finally, conclusions are offered in Section 5.

2. Related Tools

Most open-source systems adopt an Issue Tracking System (ITS) to support their maintenance processes. Normally, in such systems both users and testers can report modification requests [Liu et al. 2012]. This practice usually results in a continuous maintenance process where maintainers address the change requests as soon as possible. The ITS provides a central knowledge repository which also serves as a communication channel for geographically distributed developers and users [Anvik et al. 2006; Ihara et al. 2009].

Recent studies have focused on finding duplicated issue reports in bug tracking systems. Duplicated reports can hamper the bug triaging process and may drain maintenance resources [Cavalcanti et al. 2013]. Typically, studies for finding duplicated issues rely on traditional information retrieval techniques such as natural language processing, vector space model, and cosine similarity [Alipour et al. 2013].

Approaches to infer the most suitable developer to correct a software issue are also reported in the literature. Most of them can be viewed as recommendation systems that suggest developers to handle a reported bug. For instance, [Anvik and Murphy 2011] proposed an approach based on supervised machine learning that requires training to create a classifier. This classifier assigns the data (bug reports) to the closest developer.

However, to the best of our knowledge, we are not aware of any tool designed to recommend similar bugs to maintainers of open-source systems.

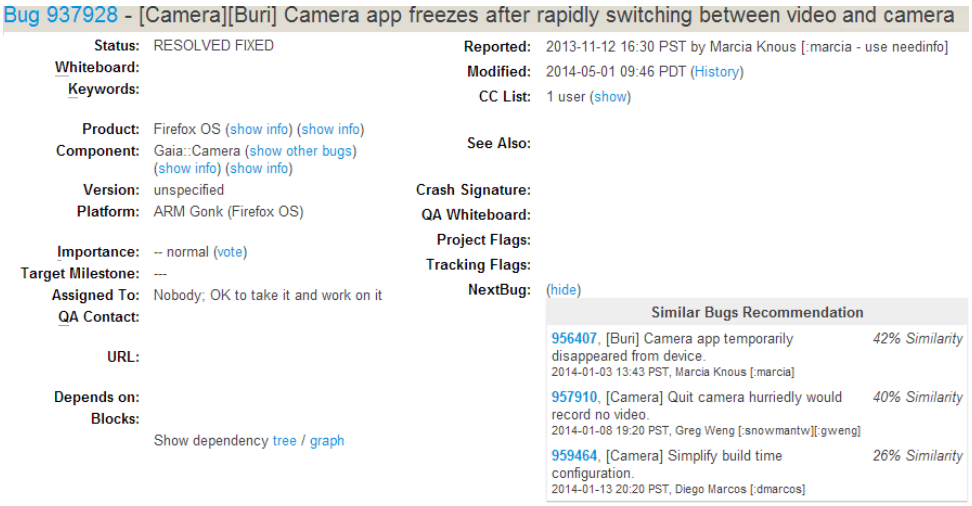

Figure 1. NextBug Screenshot (similar bugs are shown in the lower right corner)

3. NextBug in a Nutshell

In this section, we present NextBug[1] main features (Section 3.1). We also present the tool's architecture and main components (Section 3.2).

3.1. Main Features

Currently, there are several ITSs that are used in software maintenance such as Bugzilla, Jira, Mantis, and RedMine. NextBug was implemented as a Bugzilla plug-in mainly because this ITS is used by the Mozilla project, which was used to validate our tool.

When a developer is analyzing or browsing an issue, NextBug can recommend similar bugs in the usual Bugzilla web interface. As described in Section 3.2, NextBug uses a textual similarity algorithm to verify the similarity among bug reports.

Figure 1 shows an usage example of our tool. This figure shows a real bug from the Mozilla project, which refers to a FirefoxOS application issue related to a mobile device camera (Bug 937928). As we can observe, Bugzilla shows detailed information about this bug, such as a summary description, creation date, product, component, operational system, and hardware information. NextBug extends this original interface by showing a list of similar bugs to the browsed one. This list is shown on the lower right corner. Another important feature is that NextBug is only executed if its Ajax link is clicked and, thus, it will not cause additional overhead or hinder performance to developers who do not wish to use similar bug recommendations.

In Figure 1, NextBug suggested three similar bugs to the one which is browsed on the screenshot. As we can note, NextBug not only detects similar bugs but it also calculates an index to express this similarity. Our final goal is to guide the developer's workflow by suggesting similar bugs to the one he/she is currently browsing. If a developer chooses to handle one of the recommended bugs, we claim he/she can minimize the context change inherent to the task of handling different bugs and, consequently, improve his/her productivity.

[1]NextBug is open-source and available under the Mozilla Public License (MPL) at <http://aserg.labsoft.dcc.ufmg.br/nextbug/>.

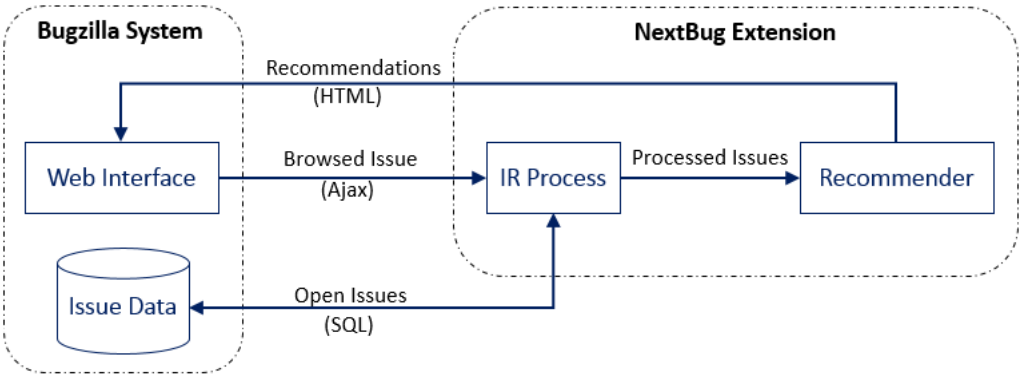

Figure 2. NextBug Architecture

3.2. Architecture and Algorithms

Figure 2 shows NextBug's architecture, including the system main components and the interaction among them. As described in Section 3.1, NextBug is a plug-in for Bugzilla. Therefore, it is implemented in Perl, the same language used in the implementation of Bugzilla. Basically, NextBug instruments the Bugzilla interface used for browsing and for selecting bugs reported for a system. NextBug registers an Ajax event in this interface that calls NextBug passing the browsed issue as an input parameter.

NextBug architecture has two central components: *Information Retrieval (IR) Process* and *Recommender*. The *IR Process* component obtains all open issues currently available on the Bugzilla system along with the browsed issue. Then it relies on IR techniques for natural language processing such as: tokenization, stemming, and stop-words removal [Runeson et al. 2007]. We implemnted all such techniques in Perl. After this processing, the issues are transformed into vectors using the vector space model [Baeza-Yates and Ribeiro-Neto 1999; Runeson et al. 2007]. VSM is a classical information retrieval model to process documents and to quantify their similarities. The usage of VSM is accomplished by decomposing the data (available bug reports and queries) into t-dimensional vectors, assigning weights to each indexed term. The weights w_i are positive real numbers that represent the *i*-th index term in the vector. To calculate w_i we used the following equation, which is called a *tf-idf* weight formula:

$$w_i = (1 + \log_2 f_i) \times \log_2 \frac{N}{n_i}$$

where f_i is the frequency of the *i*-th term in the document, N is the total number of documents, and n_i is the number of documents in which the *i*-th term occurs.

The *Recommender* component receives the processed issues and verifies the ones similar to the browsed issue. The similarity is computed using the cosine similarity measure [Baeza-Yates and Ribeiro-Neto 1999; Runeson et al. 2007]. More specifically, the similarity between the vectors of a document d_j and a query q is described by the following equation, which is called the cosine similarity because it measures the cosine of the angle between the two vectors:

$$Sim(d_j, q) = \cos(\Theta) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{||\overrightarrow{d_j}|| \times ||\overrightarrow{q}||} = \frac{\sum_{i=1}^{t} w_{i,d} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} (w_{i,d})^2} \times \sqrt{\sum_{i=1}^{t} (w_{i,q})^2}}$$

Since all the weights are greater or equal to zero, we have $0 \leq Sim(d_j, q) \leq 1$, where zero indicates that there is no relation between the two vectors, and one indicates the highest possible similarity, i.e., both vectors are actually the same.

The issues are then ordered according to their similarity before being returned to Bugzilla. Since NextBug started as an Ajax event, the recommendations are showed in the same Bugzilla interface used by developers for browsing and selecting bugs to fix.

4. Evaluation

We used a dataset with bugs from the Mozilla project to evaluate the proposed tool. Mozilla is composed of 69 products from different domains which are implemented in different programming languages. Mozilla project includes some popular systems such as Firefox, Thunderbird, SeaMonkey, and Bugzilla. We considered only issues that were actually fixed from January 2009 to October 2012. More specifically, we ignored issue types such as "duplicated", "incomplete", and "invalid".

Mozilla issues are also classified according to their severity in the following scale: blocker, critical, major, normal, minor, and trivial. Table 1 shows the number and the percentage of each of these severity categories in the considered dataset. This scale also includes enhancements as a particular severity category. Although, it was not considered in our study, i.e., we do not provide recommendations for similar enhancements.

Table 1. Issues per Severity

Severity	Issues		Days to Resolve				
	Number	%	Min	Max	Avg	Dev	Med
blocker	2,720	2.08	0	814	15.44	52.25	1
critical	7,513	5.76	0	1258	37.87	99.52	6
enhancement	3,600	2.76	0	1285	126.14	195.25	40
major	7,508	5.75	0	1275	41.59	109.83	5
minor	3,660	2.80	0	1355	77.05	161.72	11
normal	103,385	79.23	0	1373	46.27	108.84	8
trivial	2,109	1.62	0	1288	80.84	164.74	11
Total	130,495	100	–	–	–	–	–
Final Dataset	109,145	83.64	–	–	–	–	–

Table 1 also shows the number of days required to fix the issues in each category. We can observe that *blocker* bugs are quickly corrected by developers, showing the lowest values for maximum, average, standard deviation, and median measures among the considered categories. The presented lifetimes also indicate that issues with *critical* and *major* severity are closer to each other. Finally, *enhancements* are very different from the others, showing the highest values for average, standard deviation, and median.

Issues marked as blocker, critical, or major were not considered in our evaluation because developers have to fix them as quickly as possible. In other words, they would probably not consider fixing other issues together, since their ultimate priority is to fix the main *blocker* issue. In other words, our dataset is formed by fixed issues classified as normal, minor, and trivial. These issues count for 109,154 bugs (83.64%) from our initial population of bugs available for the NextBug evaluation.

We used three metrics in our evaluation: Feedback, Precision, and Likelihood. These metrics were inspired by the evaluation followed by the ROSE recommendation system [Zimmermann et al. 2004]. Feedback presents the ratio of queries where NextBug makes at least k recommendations. Precision indicates the percentage of recommendations that were actually relevant among the top-k suggestions by NextBug. Finally, Likelihood indicates whether at least one relevant recommendation is included in NextBug's top-k suggestions.

In this evaluation, we defined a relevant recommendation as one that shares the same developer with the main issue. More specifically, we consider that a recently created issue q is connected to a second opened issue when they are handled by the same developer. The assumption in this case is that our approach fosters gains of productivity whenever it recommends issues that were later fixed anyway by the same developer.

Figure 3 shows the average feedback (left chart), precision (central chart) and likelihood (right chart) up to $k = 5$.

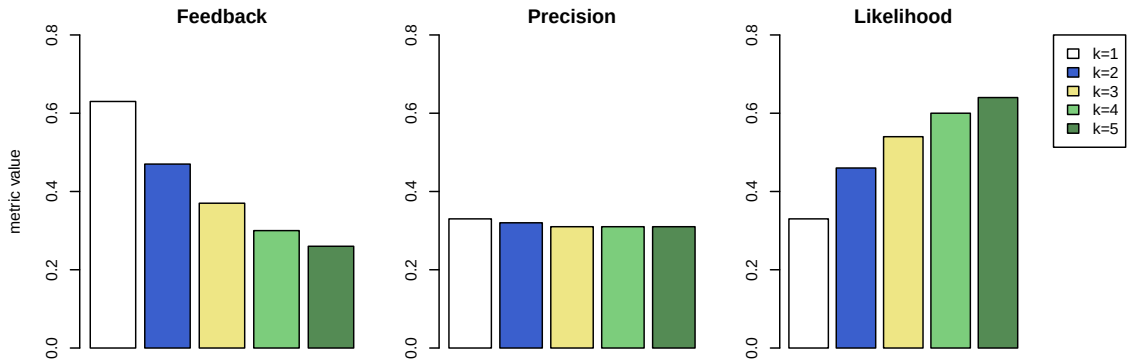

Figure 3. Average Evaluation Results for $k = 1$ to $k = 5$.

We summarize our results as follows:

- We achieved a feedback of 0.63 for $k = 1$. Therefore, on average, NextBug made at least one suggestion for 63% of the bugs, i.e., for every five bugs NextBug was able to provide at least one similar recommendation for three of those. Moreover, NextBug showed on average 3.2 recommendations for its queries.
- We achieved a precision of 0.31 or more for all values of k. In other words, the NextBug recommendations were on average 31% relevant (i.e., further handled by the same developer), no matter how many suggestions were given.
- We achieved a likelihood of 0.54 for $k = 3$. More specifically, in about 54% of the cases, there is a top-3 recommendation that was later handled by the same developer responsible for the original bug.

We also conducted a survey with Mozilla developers using our tool. We gave recommendations suggested by NextBug to 176 Mozilla maintainers and asked them a few questions. Our summarized results are: (i) 77% found our recommendations relevant; (ii) 85% confirmed that a tool to recommend similar bugs would be useful to the Mozilla community and it would allow them to do more work in less time.

4.1. Example of Recommendation

Table 2 presents an example of a bug (browsed or main issue) opened for the component `DOM:Device Interfaces` of the `Core` Mozilla product and the first three recommendations (top-3) suggested by our tool for this bug. As we can observe in the summary description, both query and recommendations require maintenance in the `Device Storage` API, used by Web apps to access local file systems. Moreover, all four issues were handled by the same developer (Dev ID 302291).

Table 2. Example of Recommendation

	Similarity	Bug ID	Summary	Creation Date	Fix Date
Browsed	–	788588	Device Storage - Default location for device storage on windows should be NS_WIN_PERSONAL_DIR	2012-09-05	2012-09-06
Top-1	56%	754350	Device Storage - Clean up error strings	2012-05-11	2012-10-17
Top-2	47%	788268	Device Storage - Convert tests to use public types	2012-09-04	2012-09-06
Top-3	42%	786922	Device Storage - use a properties file instead of the mime service	2012-08-29	2012-09-06

We can also observe that the three recommended issues were created before the original query. In fact, the developer fixed the bugs associated to the second and the third recommendations in the same date which he has fixed the original query, i.e. on 2012-09-06. However, he only resolved the other recommended bug (ID 754350) 41 days latter, i.e., on 2012-10-17. Therefore, our approach would have helped this maintainer to discover quickly the related issues. This task probably demanded more effort without a recommendation automatically provided by a tool like NextBug.

5. Conclusion

This paper presented NextBug, a tool for recommending similar bugs. NextBug is implemented as a plug-in for Bugzilla, a widely used Issue Tracking Systems (ITS), specially used by open-source systems. The proposed tool relies on information retrieval techniques to extract semantic information from issue reports in order to identify the similarity of open bugs with the one that is being handled by a developer.

We evaluate the NextBug with a dataset of 109,154 Mozilla bugs, achieving feedback results of 63%, precision results around 31% and likelihood results greater than 54%. Those results are very reasonable compared to other recommendation tools.

We also conducted a survey with 176 Mozilla developers using recommendations provided by NextBug. From such developers, 77% of them thought our recommendations were relevant and 85% confirmed that a tool like NextBug would be useful to the Mozilla community.

6. Acknowledgements

This work was supported by CNPq, CAPES, and FAPEMIG.

References

[Alipour et al. 2013] Alipour, A., Hindle, A., and Stroulia, E. (2013). A contextual approach towards more accurate duplicate bug report detection. In *10th Working Conference on Mining Software Repositories (MSR)*, pages 183–192.

[Anvik et al. 2006] Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *28th International Conference on Software engineering (ICSE)*, pages 361–370.

[Anvik and Murphy 2011] Anvik, J. and Murphy, G. C. (2011). Reducing the effort of bug report triage: recommenders for development-oriented decisions. *ACM Transactions on Software Engineering Methodology (TOSEM)*, 20(3):10:1–10:35.

[Aziz et al. 2009] Aziz, J., Ahmed, F., and Laghari, M. (2009). Empirical analysis of team and application size on software maintenance and support activities. In *1st International Conference on Information Management and Engineering (ICIME)*, pages 47–51.

[Baeza-Yates and Ribeiro-Neto 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). *Modern information retrieval*. Addison-Wesley, 2nd edition.

[Banker and Slaughter 1997] Banker, R. D. and Slaughter, S. A. (1997). A field study of scale economies in software maintenance. *Management Science*, 43:1709–1725.

[Cavalcanti et al. 2013] Cavalcanti, Y. C., Mota Silveira Neto, P. A., Lucrédio, D., Vale, T., Almeida, E. S., and Lemos Meira, S. R. (2013). The bug report duplication problem: an exploratory study. *Software Quality Journal*, 21(1):39–66.

[Ihara et al. 2009] Ihara, A., Ohira, M., and Matsumoto, K. (2009). An analysis method for improving a bug modification process in open source software development. In *7th International Workshop Principles of Software Evolution and Software Evolution (IWPSE-Evol)*, pages 135–144.

[Junio et al. 2011] Junio, G., Malta, M., de Almeida Mossri, H., Marques-Neto, H., and Valente, M. (2011). On the benefits of planning and grouping software maintenance requests. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 55–64.

[Liu et al. 2012] Liu, K., Tan, H. B. K., and Chandramohan, M. (2012). Has this bug been reported? In *20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, pages 28:1–28:4.

[Marques-Neto et al. 2013] Marques-Neto, H., Aparecido, G. J., and Valente, M. T. (2013). A quantitative approach for evaluating software maintenance services. In *28th ACM Symposium on Applied Computing (SAC)*, pages 1068–1073.

[Mockus et al. 2002] Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346.

[Runeson et al. 2007] Runeson, P., Alexandersson, M., and Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE)*, pages 499–510.

[Tan and Mookerjee 2005] Tan, Y. and Mookerjee, V. (2005). Comparing uniform and flexible policies for software maintenance and replacement. *IEEE Transactions on Software Engineering*, 31(3):238–255.

[Zimmermann et al. 2004] Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A. (2004). Mining version histories to guide software changes. In *26th International Conference on Software Engineering (ICSE)*, pages 563–572.