Coping with an Open Bug Repository

John Anvik, Lyndon Hiew and Gail C. Murphy
Department of Computer Science
University of British Columbia
{janvik, lyndonh, murphy}@cs.ubc.ca

ABSTRACT

Most open source software development projects include an open bug repository—one to which users of the software can gain full access—that is used to report and track problems with, and potential enhancements to, the software system. There are several potential advantages to the use of an open bug repository: more problems with the system might be identified because of the relative ease of reporting bugs, more problems might be fixed because more developers might engage in problem solving, and developers and users can engage in focused conversations about the bugs, allowing users input into the direction of the system. However, there are also some potential disadvantages such as the possibility that developers must process irrelevant bugs that reduce their productivity. Despite the rise in use of open bug repositories, there is little data about what is stored inside these repositories and how they are used. In this paper, we provide an initial characterization of two open bug repositories from the Eclipse and Firefox projects, describe the duplicate bug and bug triage problems that arise with these open bug repositories, and discuss how we are applying machine learning technology to help automate these processes.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

General Terms

Management

Keywords

Bugzilla, duplicate detection, triage, machine learning

1. INTRODUCTION

Proponents of open source software development believe that allowing the users of the software to easily report, and sometimes help fix, bugs improves the quality of the software produced [6, 7]. To enable these contributions by users, most open source projects provide an open bug repository (e.g., Bugzilla[1]). These repositories typically provide a means of categorizing, describing and tracking both problems with, and potential enhancements to, the software system. They provide a means for the developers and users to engage in focused, archived conversations about the system.

Although open source software developers interact with the bug repository often, there is little data available characterizing their interactions (Section 2). In this paper, we present data to fill this gap, providing a characterization of the data in and the use of parts of the bug repositories for two open source projects (Section 4): Eclipse (V3.0) and Firefox (V1.0). This data confirms two problems that arise with open bug repositories that open source developers have communicated to us previously (Section 5): the difficulty of detecting which bug reports are duplicates of those already in the repository (the duplicate bug problem), and the difficulty of assigning new bug reports to the appropriate developer (the bug triage problem [8]). Currently, the approaches taken to these problems are human-oriented; humans must read the bugs and decide upon whether they are duplicates, and to whom they should be assigned. We believe these processes can, at least in part, be automated by using the historical information about the bug processes stored in the bug repository. We have been investigating the use of machine learning technology to this purpose and report on some of our initial results (Section 6).

The data and ideas presented in this paper provide a basis for considering the kind of support that should be integrated into a development environment to support better bug reporting and tracking.

2. RELATED WORK

Information stored in a bug repository has been used by a number of researchers to investigate questions about the processes used in open source development. Mockus, Fielding and Herbsleb use bug information to gauge the various roles people played in the Apache and Mozilla open source projects [6]. Crowston and Howison use bug repository information to analyze the social structure of a number of open source projects [3]. Other projects have used information in the bug repository to help automate bug assignment [1][2][4]. In comparison, this paper focuses solely on characterizing what information is in a bug repository and how it is used.

[1] `www.bugzilla.org` as verified 12/08/05

Sandusky and colleagues also focus solely on the information in the bug repository. The goal of their analysis was to identify bug report networks, which are groupings of bug reports due to duplication, dependency or reference relationships described in the bug reports as a means of improving how problems are managed [10]. We too are addressing the management of bugs but we are using different analyses and techniques.

3. ANATOMY OF A BUG REPORT

Both of the projects we consider in this paper use Bugzilla as their open bug repository. A bug report in Bugzilla has several parts: pre-defined fields, free-form text, attachments, and dependencies.

The pre-defined fields provide a variety of categorical data about the bug report. Some values, such as the report identification number and reporter, are fixed when the report is created. Other values, such as the product, component, operating system, version, priority, and severity, are selected by the reporter when the report is filed, but may also be changed over the lifetime of the report. Other fields routinely change over time, such as the person to whom the report is assigned, the current status of the report, and if resolved, its resolution state. There is also a list of the emails of people who have asked to be kept up to date on the activity of the bug.

A Bugzilla bug report has three free-form parts: a one-line summary of the bug, a full description of the bug, and additional comments. The one-line summary becomes the title of the report. The full description contains an elaborated description of the effects of the bug and the necessary information for a developer to reproduce the bug. The additional comments contain any comments made by the public and developers such as a discussion of possible fixes, or when another bug report is marked as a duplicate of this report.

Reporters and developers may provide attachments to reports. Attachments provide additional information about the bug, such as a screenshot of the erroneous behaviour.

Bugzilla also tracks two other pieces of information. First, it tracks which bugs block the resolution of other bugs. Second, it tracks activity performed on each bug, forming a log that provides a history of how the report has changed over time, such as when the report has been reassigned, or when its priority has been changed.

4. HOW ARE BUG REPORTS USED?

To investigate how open bug repositories are used, we analyzed the use of the repositories for the Eclipse (V3.0) and Firefox (V1.0) projects. These versions were chosen as they represent relatively mature offerings of each of these systems.

The Eclipse dataset totaled 18,165 reports created in the time period of October 2001 to August 2005. The Firefox dataset contained 2,013 reports created from May 2003 to August 2005. Table 1 shows the dataset broken down by the year in which the reports were created. For both projects there are a few bugs that were carried over to the project from previous releases. As Eclipse V3.0 was released in June 2004, with its first milestone in June 2003, the bulk of the reports were created in this time frame. For Firefox, V1.0 was released at the beginning of November 2004, and likewise the bulk of reports are from that time period.

Table 1: Bug Reports by Year

	2001	2002	2003	2004	2005
Eclipse	2	13	3,912	13,130	1,108
Firefox	-	-	7	1,194	812

4.1 What Kind of Reports Are Stored?

To determine the kinds of bug reports in the repositories, we categorized the reports by their status.

When a bug report is submitted its status is set to either NEW or UNCONFIRMED, depending on the conventions of the project. Once a developer has been either assigned to or accepted responsibility for the report, the status is set to ASSIGNED.

When a bug report is closed its status is set to RESOLVED. It may further be marked as being verified by a quality assurance group (VERIFIED) or closed for good (CLOSED). A report can be resolved in a number of ways and Bugzilla bug reports indicate this with a resolution status. If the resolution resulted in a change to the code base, the bug is resolved as FIXED. When a developer determines that the report is a duplicate of an existing report then it is marked as DUPLICATE. If the developer was unable to reproduce the bug it is indicated by setting the resolution status to WORKSFORME. If the report describes a problem that will not be fixed, is not an actual bug, or is tracked in another repository, the report is marked as WONTFIX, INVALID, or MOVED respectively. A formerly resolved report may be reopened at a later date, and will have its status set to REOPENED to indicate this.[2]

Figure 1 shows the proportion of bug reports with each status type for the two projects. The reports with a status of UNCONFIRMED, NEW, ASSIGNED, or REOPENED have been collected under the heading of OPEN. Interestingly, the proportion of OPEN and FIXED bugs, bugs that could lead to actual changes to the system, for Eclipse is only 58% and for Firefox is 44%. Figure 1 also shows that the bug repository is used to track a number other items.

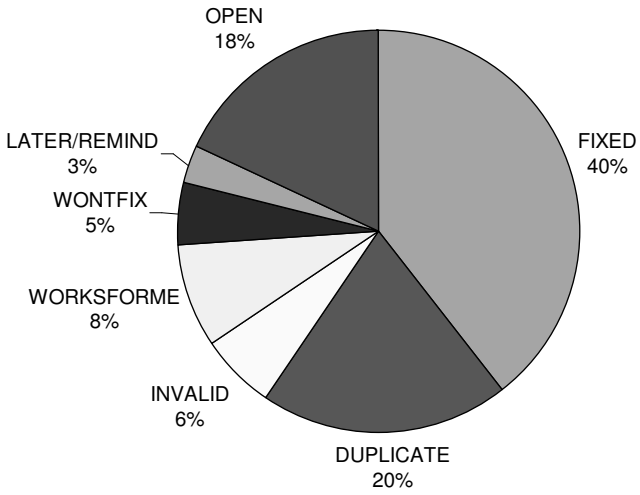
4.2 Who is Reporting and at What Rate?

To understand how many reports must be processed by developers, we considered how many new reports are filed daily.

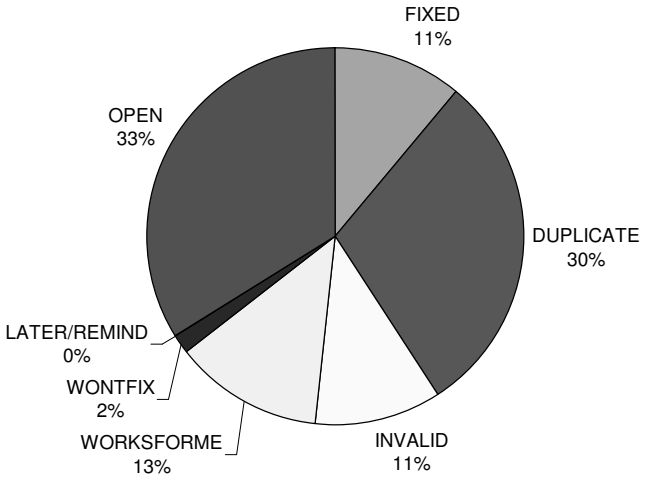
Table 2 shows how often new reports were submitted in the three months prior-to and after the release of the two projects,[3] and how often new reports have been submitted since the release of each project. As one would expect, the

[2]The complete bug report life-cycle supported by Bugzilla is found at `http://www.bugzilla.org/docs/tip/html/lifecycle.html` (as verfied 29/09/05)

[3]Eclipse 3.0 was released June 25, 2004 and Firefox 1.0 was released November 9, 2004



(a) Types of bugs for Eclipse.



(b) Types of bugs for Firefox.

Figure 1: Proportion of bug types for Eclipse and Firefox.

Table 2: Number of Bugs Submitted Daily.

	Around Release		After Release	
	Eclipse	Firefox	Eclipse	Firefox
Min	1	1	1	1
Average	48	8	13	5
Max	192	37	124	37

Table 3: Top five domains of bug report reporters.

Eclipse		Firefox	
Suffix	%	Suffix	%
ca.ibm.com	33	gmail.com	16
us.ibm.com	11	yahoo.com	9
ch.ibm.com	8	hotmail.com	5
yahoo.com	2	mozilla.org	3
gmail.com	2	iwaruna.com	2
Unique: 3268		Unique: 1500	

average number of bugs submitted daily increased around the release of a project.

Submitting reports to an open bug repository requires the reporter to establish an account. Consequently, it requires some motivation on the part of the reporter in order to report bugs. For Bugzilla, the account name is the email address of the account holder.

To understand who is submitting reports, we analyzed the email addresses of the reporters. Table 3 shows the top five domains from which bug reports were filed. Although we did not determine how many of the addresses refer to the same person, such cases are not likely to have a significant effect given the number of unique addresses.

4.3 How Quickly Are Reports Resolved?

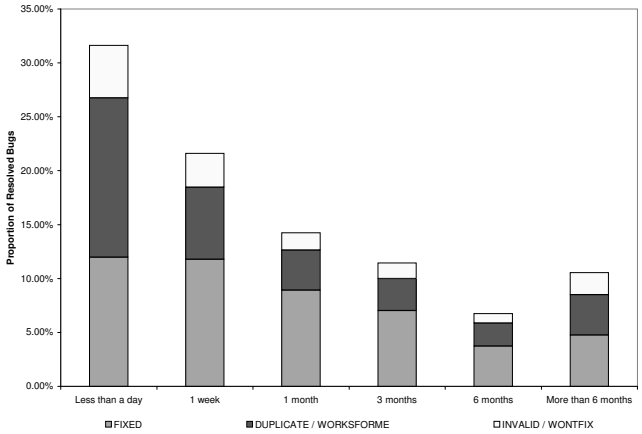
Figure 2 shows how quickly bugs get resolved for each of the projects. The figure show a couple of interesting points about each project. First, for Firefox, a great number of DUPLICATE/WORKSFORME reports are identified in less than a day. Second, for Eclipse, roughly the same number of reports are resolved as FIXED or DUPLICATE/WORKSFORME during the first day.

4.4 Who Resolves Reports?

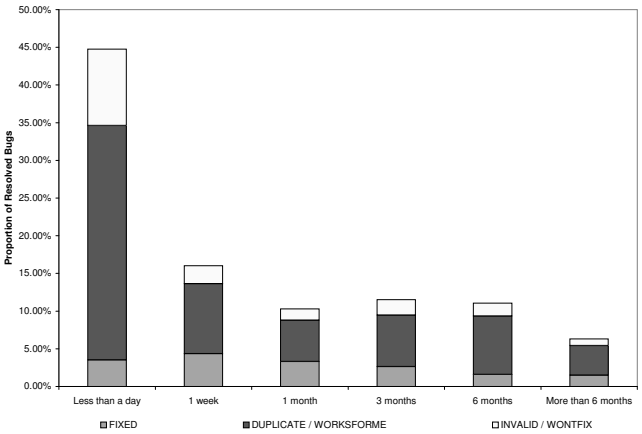
Determining the developer who resolved a report is non-trivial. One would think that the 'assigned-to' field mentioned in Section 4 would provide the information; however we have found that this is rarely the case. For example, the value of the assigned field may not refer to a specific developer. For both the Eclipse and Firefox repositories, reports are first assigned to a default email address before they are assigned to an actual developer. For trivial fixes the assigned-to field is unlikely to be changed. Alternatively, a project may have a process whereby once the developer has fixed the bug, they assign it to a quality assurance person to verify the fix.

To determine the developer who resolved a report, we examined the bug activity log and determined who resolved the report through the use of heuristics. Examples of heuristics that we used are:

1. If a report is resolved as FIXED, it was fixed by whoever submitted the last patch that was approved. (Firefox)
2. If a report is resolved as FIXED, it was fixed by whoever marked the report as resolved. (Eclipse)
3. If a report is resolved as a DUPLICATE, it was resolved by whoever resolved the report of which this is a duplicate of. (Eclipse and Firefox)



(a) The age of resolved bugs for Eclipse.



(b) The age of resolved bugs for Firefox.

Figure 2: Age of resolved bugs for Eclipse and Firefox.

Table 4: Top five domains of bug report fixers

Eclipse		Firefox	
Suffix	%	Suffix	%
ca.ibm.com	69	gmail.com	6
us.ibm.com	7	mozilla.org	2
ch.ibm.com	6	web.de	2
eclipse.org	5	pobox.com	2
magma.ca	4	steelgryphon.com	2
Unique: 234		Unique: 192	

4. If a report is resolved as WORKSFORME, it was marked as such by the person doing the bug assignment, so it is unclear who the developer would have been. The the report is labeled as unclassifiable. (Firefox)

Similar to the analysis we performed for who submits bugs (Section 4.2), we determine the top five domains of developers who resolved bugs. Table 4 shows these five domains. As one would expect, the IBM and Eclipse domains are the top domains of developers resolving bugs for the Eclipse project. Notice how the top three domains for the Eclipse project are the same as those for reporters. In contrast, the domains for reporters and resolvers for Firefox are more varied.

5. CHALLENGES RAISED BY AN OPEN BUG REPOSITORY

The data from Section 4 illustrates a number of challenges faced by developers of projects having a bug repository that is accessible to the public. This data elucidates two challenges.

1. Many reports are added daily to bug repositories. As shown in Tables **??** and **??**, Firefox receives an average of 8 bugs a day around release time and peaks with 37 in a day. The Eclipse project receives a large number of reports, averaging 48 bugs each day with a maximum of 192, around the time of a release.

2. In many cases, the time and effort used examining a report is wasted. If we consider the reports in Figure 1 that are labeled OPEN or FIXED to make a contribution to the overall quality of the product, then for Eclipse 39% of the reports do not contribute to improving the product. For Firefox, 56% of the reports do not help the project. If one further imagines that it takes a developer an average of two minutes to read a bug report, then nearly 200 person-hours were spent on the Eclipse project handling unproductive reports.

One way to reduce the time and effort spent on processing reports is to focus the developers' time on beneficial reports. As demonstrated by Figure 1, most reports that are submitted are either a duplicate of an existing report (DUPLICATE), do not describe an actual bug (INVALID), cannot be reproduced (WORKSFORME), or describes some behaviour that the developers have decided not to change (WONTFIX). Of these four categories, duplicates account for the largest percentage in both the Eclipse and Firefox projects. To identify a duplicate, the bug assigner must possess a good knowledge of past bugs, or search through past reports, to determine if a previous report exists. The need for duplicate detection was expressed by a Mozilla developer commenting that, "It's essential that duplicates be marked without developers having to look at them, there are just so many."[4]

Each submitted report requires someone to examine and determine if it details a problem that needs further investigation. Bug triage is the task of processing a bug report, which includes deciding who should be assigned the responsibility of investigating it further, and perhaps fixing it. In communication with a Mozilla triager, he said that "Everyday, almost 300 bugs appear that need triaging. This is far too

[4]Personal communication, March 1, 2005

much for only the Mozilla programmers to handle."[5] The Eclipse project previously has a single person doing triage, but due to the increasing amount of time that it took, the component teams were assigned the responsibility.[6] In addition, the task of assigning a bug to a developer becomes more burdensome, considering that in Table 4, a triager has many possibilities with 234 developers fixing Eclipse bugs and 192 developers fixing Firefox bugs (Table 4).

6. IMPROVING BUG MANAGEMENT

An avenue for addressing these challenges is to explore automatic techniques for performing triage and duplicate recognition. Coping with these challenges is similar to dealing with email. For instance, email is often categorized according to whether or not it is spam, who it is from, and the ongoing project to which it relates. For bug repositories, developers would like to be able to filter out the productive reports from those that are not helpful. Cubranic and Murphy performed initial investigations into the use of a machine learning approach to aid bug management [4]. We are continuing this investigation and are finding that machine learning can provide significant assistance in performing triage and duplicate detection.

Human bug triagers use a lot of contextual knowledge to decide to whom a new report is to be assigned. As was previously shown, the human bug triager must, particularly on some days, deal with a large number of reports. By looking at the history of who resolved previous reports, one can use machine-learning techniques to automatically predict who should be assigned a new report. We have found that by training a Support Vector Machine [5] classifier with eight months of data from the Eclipse project, the person who actually resolved a report can be correctly suggested 57% of the time.

Our approach to duplicate detection uses a statistical model built from the knowledge of past reports using machine-learning techniques. Since bugs are continually added to the repository, an incremental approach is employed to update the statistical model after seeing each new bug. This allows the model to detect duplicates of new bug reports and adapt to the changing composition of bugs in the repository. By using cosine similarity [9], the model classifies new bug reports as either being unique or duplicate. If a bug is classified as a duplicate, the three most similar existing bugs are retrieved. Using the bugs in Firefox 1.0, unique bugs are correctly identified 90% of the time and duplicate bugs are correctly identified 28% of the time, within the top three suggestions.

Automated bug triage and duplicate detection can be incorporated into the development environment to assist in bug management. These techniques can be implemented on the client side in an IDE or in the bug repository access tool (often web-based) itself. When a reporter creates a new bug report, automated duplicate detection can suggest similar existing bugs, which the reporter can then use to determine if their bug has been already reported. For each bug that has passed duplicate detection, automatic triage can suggest possible assignees to a triager, to assist in the assignment of bugs.

[5]Personal communication, March 5, 2005
[6]Personal communication, Feb 23, 2005

7. SUMMARY

Open bug repositories pose some new challenges for software development teams. We have analyzed data from the Eclipse and Firefox repositories to demonstrate two challenges: developers must deal with large numbers of reports, and many of these reports are not productive reports, rather the reports end up being marked as WORKSFORME, INVALID or DUPLICATE. To aid developers in better managing an open bug repository we have been investigating the use of machine learning approaches to automatically suggest when a new bug may be a duplicate of reports already in the repository, and to automatically suggest to whom a bug should be assigned.

8. ACKNOWLEDGMENTS

This research was funded in part by an IBM Eclipse Innovation Grant.

9. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? Unpublished. Available from the authors.

[2] G. Canfora and L. Cerulo. How software repositories can help in resolving a new change request. In *Workshop on Empirical Studies in Reverse Engineering*, September 2005.

[3] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10, 2005.

[4] D. Cubranic and G. C. Murphy. Automatic bug triage using text classification. In *Proc. of Software Engineering and Knowledge Engineering*, pages 92–97, 2004.

[5] S. R. Gunn. Support Vector Machines for classification and regression. Technical report, University of Southampton, Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science, May 1998.

[6] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, 2002.

[7] E. S. Raymond. The cathedral and the bazaar. *First Monday*, 3(3), 1998.

[8] C. Reis, R. Pontin, and M. Fortes. An overview of the software engineering process and tools in the mozilla project. In *Proc. of Open Source Soft. Dev. Workshop, Newcastle upon Tyne*, pages 155–175, 2002.

[9] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[10] R. Sandusky, L. Gasser, and G. Ripoche. Bug report networks: Varieties, strategies, and impacts in a f/oss development community. *Proc. of 1st Int'l Workshop on Mining Software Repositories*, pages 80–84, 2004.