A Comparative Exploration of FreeBSD Bug Lifetimes

Gargi Bougie
University of Victoria
gbougie@uvic.ca

Christoph Treude
University of Victoria
ctreude@uvic.ca

Daniel M. German
University of Victoria
dmg@uvic.ca

Margaret-Anne Storey
University of Victoria
mstorey@uvic.ca

***Abstract*—In this paper, we explore the viability of mining the basic data provided in bug repositories to predict bug lifetimes. We follow the method of Lucas D. Panjer as described in his paper, Predicting Eclipse Bug Lifetimes. However, in place of Eclipse data, the FreeBSD bug repository is used. We compare the predictive accuracy of five different classification algorithms applied to the two data sets. In addition, we propose future work on whether there is a more informative way of classifying bugs than is considered by current bug tracking systems.**

***Index Terms*—Mining Software Repositories; Bug lifetimes; FreeBSD;**

I. Introduction

Accurately predicting the lifetime of bugs in a software project is useful from both a management and development perspective. Allocation of resources and project schedules are often adjusted based on the number and severity of bugs and the developers' estimated time to fix. If a method for predicting the lifetime of bugs in a software project can be derived, managers and developers can better understand the scope of the work required, achieving improved accuracy of release schedules and budget plans. In addition, if the measure of bug lifetimes suggests a classification scheme other than that provided in common bug tracking systems, this may have implications for improving bug tracking methods in today's large-scale software projects.

As Juristo et al. [1] explain, replication studies are relatively rare in the field of software engineering due to the difficulty of reproducing exactly the original experimental conditions. Juristo et al. conclude, however, that valuable knowledge can be obtained from non-exact replications if the deviations are controlled and the results are analyzed and compared to those of the original experiment.

In this research, we perform a replication of the work conducted by Panjer in his evaluation of bug lifetimes in the Eclipse project [2]. In place of Eclipse Bugzilla data, we use the FreeBSD bug repository. We examine the ability to predict bug lifetimes based on historical data contained in the FreeBSD bug repository and compare the predictive accuracy of several classification algorithms for both data sets. Also considered is the resulting classification of FreeBSD bugs based on their lifetimes. Since the FreeBSD data set is large, data mining tools and techniques are used.

The literature search conducted by Panjer showed that there has been little research done in the area of predicting bug lifetimes. However, characterizing classes of bugs in the Linux and OpenBSD kernels in terms of their average life span has been investigated by Chou et al [3]. Kim et al. [4] studied the open source projects, ArgoUML and PostgreSQL, showing that they have a median bug-fix time of about 200 days and a maximum of several years.

II. Research Questions

Panjer's goal was to determine the viability of predicting a bug's lifetime from time of arrival to time of resolution, using only the basic data available in a project's bug repository. In this paper, Panjer's method is applied to a different data set, the FreeBSD bug repository, and the same classification algorithms are applied. Unlike Eclipse which uses Bugzilla, FreeBSD uses the GNATS bug tracking system. Our research questions are:

RQ1: Can we determine whether there is a superior classification algorithm for predicting bug lifetimes?

RQ2: Can the process of classifying bugs be improved compared to current bug tracking systems by leveraging non-obvious properties such as user base or expertise required?

III. Data Set

The FreeBSD bug repository was chosen as the data set for this research because FreeBSD is a large, well-known and mature open source project as is Eclipse. The partially-parsed bug repository was made available on the 2010 Mining Software Repositories Challenge website.

A. GNATS Process

The GNATS problem report life cycle [5] begins by a problem reporter submitting a problem report (PR) and receiving a confirmation message. A committer within the FreeBSD project takes interest in the problem report and either assigns it to himself, or to another developer. Once the person assigned to the PR (the assignee) has a solution to the problem, he or she submits it in a follow-up, asking the reporter to test the fix. Once the reporter and assignee agree to a fix, a Merge From Current (MFC) countdown may be started. An MFC countdown indicates the number of days a component committed to the *current*, or development branch of a software repository must wait before being committed to the *stable* branch [6]. This value is set by the committer and must be a

minimum of three days. The countdown period allows for the component to be tested by different developers on a range of configurations. If an MFC countdown is started, the assignee leaves the problem report in the "patched" state until the countdown finishes and the component is merged to the *stable* branch, at which point, the assignee closes the PR. If an MFC is not required, the assignee simply closes the PR.

B. Data Set Preparation

The provided partially-parsed FreeBSD bug repository included GNATS messages containing message_ID, message_body and several other fields deemed irrelevant to this research. The message_body field included the following information: *bug_number, category, confidential, severity, priority, responsible, state, quarter, keywords, date_required, class, submitter_id, arrival_date, closed_date, last_modified, originator, organization, environment*. We parsed this information from the *message_body* using perl scripts and placed it in a new database table for exploration.

The data set contained 17 entries where *bug_number* was null, so we removed these. In addition, 5963 entries contained an invalid value for *arrival_date* or *closed_date*, which we also removed. Invalid values included cases where the *closed_date* was earlier than the *arrival_date* or where either of the fields were null. This left 131,613 valid entries.

The data set contains bugs opened between September 14th, 1994 and September 11th, 2009 inclusive. The maximum number of days a bug is open is 4051.96 days, the median is 4.23 days, the mean is 86.24 days, and the standard deviation is 253.97 days. From these numbers, we can see that the distribution of bug lifetimes is heavily skewed. The box plot in Figure 1 illustrates the time to close bugs in days for the FreeBSD project.

In order to apply data mining algorithms that require nominal attributes to the data set, the WEKA toolkit [7] was used to discretize the bug lifetime values in days using an equal frequency binning algorithm. We did this for seven bins, as per Panjer's method, and also for ten bins, for an additional comparison. The results for ten bins are shown in Figure 2. Panjer's Eclipse data produced one bin that was significantly larger than the others. We did not see this issue with the 7-bin or 10-bin FreeBSD data.

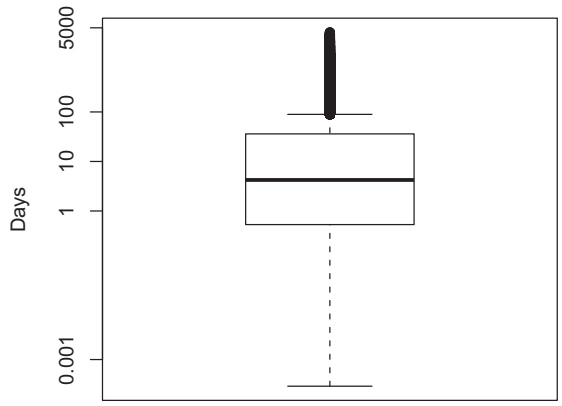

Fig. 1. Distribution of FreeBSD bug lifetimes in days (logarithmic scale)

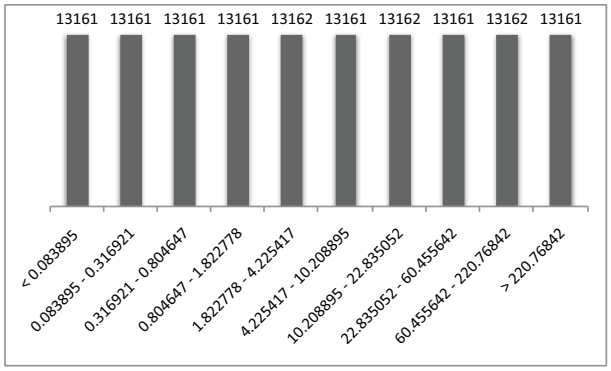

Fig. 2. Result of equal frequency binning algorithm with 10 bins. Shows number of bugs in each lifetime range (lifetime ranges in days).

IV. Automatic Classification

Once the bug lifetime values were discretized, basic data mining and analysis was performed on both the 7-bin and 10-bin data set using classification algorithms provided in WEKA. These algorithms were meant to provide a mechanism for determining the accuracy with which bug lifetimes can be predicted using the prepared data set. The basic algorithms, 0-R and 1-R were executed on the data set to establish a prediction accuracy baseline. In addition, the Naive Bayes algorithm, C4.5 Decision Trees, and Logistic Regression algorithms were applied. WEKA uses 10-fold cross-validation [7] for these algorithms. The kappa statistic is presented as a means of measuring the accuracy of the predicted classification against the actual classification. The kappa statistic offers a more robust measure than the individual percentage since it takes into account the number of correct classifications occurring by chance. Both the kappa statistic and the individual percentage of correctly predicted classifications are recorded. Interesting findings from the application of each algorithm to the data set are described. A summary of results generated from analyzing the 7-bin and 10-bin FreeBSD GNATS data is shown in Table 1 and Table 2, respectively. The results of Panjer, generated from analyzing the Eclipse Bugzilla data, are shown in Table 3. Detailed outcomes of applying the algorithms are described below for the 10-bin data set.

A. 0-R and 1-R

The 0-R algorithm correctly classifies 10.00% (14.28% with 7 bins) of bugs and the kappa statistic is 0. The 1-R algorithm builds a single-level decision tree by generating a rule for each of the supplied attributes. The attribute used in the best-

performing rule is selected for the root node of the tree. The 1-R algorithm correctly classifies 17.00% (22.95% with 7 bins) of bugs and the kappa statistic is 0.0779 (0.1011 for 7 bins). The attribute selected to classify all instances is the *category*. The rule states that bugs categorized as being related to *ports* are resolved in $1.82 < \text{days} < 4.23$, while those categorized as *ia64* are resolved in $60.46 < \text{days} < 220.77$. For the *www* category, the results show that bugs are closed within $4.23 < \text{days} < 10.21$, and for *kern, bin, docs, misc, i386, conf, usb, gnu, amd64, java, standards, alpha, sparc64*, and *threads*, bugs are closed in $220.77 < \text{days}$. Bugs labelled *junk, advocacy*, and *sun4v* are closed within $\text{days} < 0.08$.

B. Decision Trees

The J48 classifier in WEKA was used for generating C4.5 decision trees. The algorithm first generates a number of cases for the attributes provided in the numeric or nominal data set. Within these cases, the J48 algorithm looks for patterns that differentiate the attributes [8]. In our case, the C4.5 algorithm correctly classifies 19.49% (25.73% for 7 bins) of bugs and the kappa statistic is 0.1054 (0.1335 for 7 bins). As with the 1-R algorithm, the top node of the generated trees is always the *category*. The child of category was *class* in all cases except: *kern*, in which case category was followed by *confidential*; *junk, www*, and *arm*, in which cases category was followed by *severity*; and *standards*, in which case category was followed by *priority*.

C. Naive Bayes

The Naive Bayes algorithm is based on Bayes' theorem of conditional probability and assumes all attributes are independent. Naive Bayes correctly classifies 18.45% (24.99% for 7 bins) of bugs and the kappa statistic is 0.0939 (0.1249 for 7 bins).

D. Logistic Regression

As indicated by Panjer, computational constraints make processing the entire data set using logistic regression difficult. Therefore, following the method taken by Panjer, a randomly selected subset of 496 bugs, or 3.77% of the original data set was used. The regression model that is built using the logistic regression algorithm correctly classifies 16.53% (23.99% for 7 bins) of bugs and the kappa statistic is 0.0724 (0.1133 for 7 bins).

Algorithm	Predicted %	Kappa
0-R	14.28%	0.0000
1-R	22.95%	0.1011
C4.5 Decision Tree	25.73%	0.1335
Naive Bayes	24.99%	0.1249
Logistic Regression	23.99%	0.1133

TABLE I
SUMMARY OF PREDICTION RESULTS FOR FREEBSD (7 BINS)

Algorithm	Predicted %	Kappa
0-R	10.00%	0.0000
1-R	17.00%	0.0779
C4.5 Decision Tree	19.49%	0.1054
Naive Bayes	18.45%	0.0939
Logistic Regression	16.53%	0.0724

TABLE II
SUMMARY OF PREDICTION RESULTS FOR FREEBSD (10 BINS)

Algorithm	Predicted %	Kappa
0-R	29.10%	0.0000
1-R	31.00%	0.0747
C4.5 Decision Tree	31.90%	0.0938
Naive Bayes	32.50%	0.1195
Logistic Regression	34.90%	0.1577

TABLE III
SUMMARY OF PREDICTION RESULTS FOR ECLIPSE

V. DISCUSSION

0-R sets a baseline for the prediction accuracy of bug lifetimes. It simply takes the most commonly occurring class of lifetime values and makes a prediction that all bugs will be resolved within that time period.

The 1-R algorithm selects what is considered to be the most important attribute by generating the first rule that would appear in a decision tree for the data set. The selection of *category* as the root attribute in this case is logical since each category defines a unique subset of the project for which bugs may have similar resolution times. For 10 bins, 1-R generates the rule defining category $=$ *ports, junk, www, powerpc, advocacy, pending*, or *sun4v* to predict bug resolution times of $\text{days} < 10.21$. All other categories always imply resolution times of greater than 220.77 days, except *ia64* which implies a resolution time of $60.46 < \text{days} < 220.77$, and *arm* which implies a resolution time of $10.21 < \text{days} < 22.84$.

Though difficult to hypothesize a reason for this behavior without further investigation, it is not surprising that bugs labelled as *pending* and *junk* are closed more quickly than bugs in many other categories. It is also interesting to note that categories for the specific processors, *ia64* and *arm*, imply unique resolution times. Valuable future work would involve exploring the properties of bugs in more depth, especially those properties not explicitly captured by bug tracking systems. Examples may include looking at the user base size for the component a bug is found in, or determining the number of distinct developers assigned to fix bugs in the different categories. Such work may shed light on more informal practices such as fixing components with a wider user base first, or leveraging a core group of expert developers to fix bugs in a certain category quickly.

Comparing our results obtained from mining the FreeBSD bug repository to Panjer's findings for the Eclipse Bugzilla data, we can see that the prediction accuracy is higher for the Eclipse data using all algorithms. However, the kappa statistic is higher for the 10-bin FreeBSD data using the 1-

R and C4.5 Decision Tree algorithms. For the 7-bin FreeBSD data, the kappa statistic is higher than seen with Eclipse for all algorithms except Logistic Regression. It is not clear in Panjer's paper why one bin is much larger than the others. We can see, however, that his results for the correctly predicted percentage do not vary significantly from 0-R to the other algorithms. Our 10-bin results demonstrate an improvement in predictive accuracy of approximately 97% from 0-R (10.00%) to C4.5 (19.49%). This difference implies that the predictive accuracy and optimal methods for predicting bug lifetimes may vary significantly between software projects. It is apparent that no one algorithm can achieve a higher predictive accuracy across all projects. Future work would be to investigate the bug lifetimes and prediction accuracy for a range of software projects and to study their structure and organization.

VI. Conclusion

In this research, bug lifetimes for the FreeBSD software project are modeled using only the information available in the project's bug repository. The WEKA toolkit is used to execute data mining algorithms on the data set. As a result, it was determined that bug lifetimes for the FreeBSD project can be predicted with an accuracy of 19.49%, using 10 bins for lifetime ranges. The attributes of the data set most suitable for prediction were discovered to be *category*, *severity*, *priority*, and *confidentiality*. Comparing our results to Panjer's findings for the Eclipse data set reveals that the predictive accuracy of classification algorithms varies across software projects.

References

[1] N. Juristo and S. Vegas, "Using differences among replications of software engineering experiments to gain knowledge," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 356–366.

[2] L. D. Panjer, "Predicting eclipse bug lifetimes," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007, p. 29.

[3] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. ACM, 2001, pp. 73–88.

[4] S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 173–174.

[5] T. T. Dinh-Trong and J. M. Bieman, "The freebsd project: A replication case study of open source development," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 481–494, 2005.

[6] "*A Project Model for the FreeBSD Project*," *Available online at http://people.freebsd.org/~jcamou/en_US.ISO8859-1/books/dev-model/book.html*. [Online]. Available: http://people.freebsd.org/~jcamou/en_US.ISO8859-1/books/dev-model/book.html

[7] "*Weka 3 - Data Mining Software in Java*," *Available online at http://www.cs.waikato.ac.nz/ml/weka/*. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka

[8] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.