Bug Fix-Time Prediction Model Using Naïve Bayes Classifier

W. Abdelmoez, Mohamed Kholief, Fayrouz M. Elsalmy
walid.abdelmoez@aast.edu, kholief@aast.edu, fayrouz_elsalmy@hotmail.com
Arab Academy for Science, Technology, and Maritime Transport
ALEXANDRIA, EGYPT P.O.BOX1029

ABSTRACT

Predicting bug fix-time is an important issue in order to assess the software quality or to estimate the time and effort needed during the bug triaging. Previous work has proposed several bug fix-time prediction models that had taken into consideration various bug report attributes (e.g. severity , number of developers ,dependencies) in order to know which bug to fix first and how long it will take to fix it. Our aim is to distinguish the very fast and the very slow bugs in order to prioritize which bugs to start with and which to exclude at the mean time respectively. We used the data of four systems taken from three large open source projects Mozilla, Eclipse, Gnome. We used naïve Bayes classifier to compute our prediction model.

I. INTRODUCTION

Bug tracking systems, such as Bugzilla [1] [2], are used to manage and facilitate the bug resolution process in order to develop and to maintain the software systems effectively. These bug tracking systems record different features about reported bugs, such as the time the bug was reported and the component the bug was found in. The information stored in bug tracking systems is leveraged by many researchers to investigate different phenomena.

Taking into consideration the development effort coordination, two questions need to be addressed: which bug to fix first and how long it will take to fix. Providing software developers with bug categorizing recommendations helps and supports them in cost/benefit analysis. In this paper we investigate if we can categorize the incoming bugs into:

- Very fast bugs that the maintainer could start to fix them and,
- Very slow bugs in order to exclude them at the mean time especially if they do not have a high priority,

The hypotheses of our empirical studies are:

H1—

Incoming bug reports can be classified into very fast fixed bugs and not very fast fixed.

H1

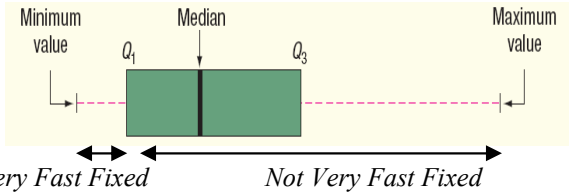

Figure 1 Graphical representation of H1

H2—

Incoming bug reports can be classified into very slowly fixed bugs and not very slowly fixed bugs.

H2

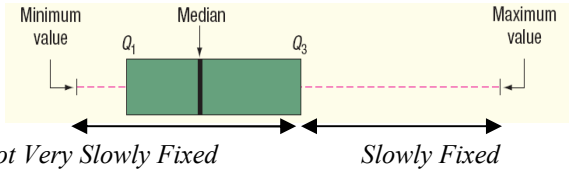

Figure 2 Graphical representation of H2

We use bug report data of four software systems taken from the three open source projects Eclipse, Mozilla, and Gnome. Naïve bayes network with 10-fold cross validation is used to train and test prediction models. The predictive power of each model is evaluated with precision and recall.

In [3], Giger et al. used the median for binning the bug reports into fast and slow. In this paper, we propose to use the first quartile Q1 to categorize the

978-1-4673-2824-1/12/$31.00 ©2012 IEEE

bugs into very fast and not very fast fixed as shown in Figure 1. Also, we propose to use the third quartile Q3 to categorize the bugs into very slowly and not very slowly fixed as shown in Figure 2

The paper is organized as follows. In Section II, we discuss the required background. Section III presents our proposed analysis approach. In Section VI, results of the experiment are examined. Section V discusses the related work. We conclude the paper and discuss the future work in Section VI.

II. BACKGROUND

In this section, we give a brief background on area that will be needed in our analysis of the bug fix-time data. As mentioned in the introduction, we will be using quartiles so in this section there is a brief on it. Also, we give a hint on using 10-fold-cross validation of prediction models. At the end, a brief introduction on naïve Bayes algorithm that we used to compute our predictive model is given.

A way to describe the spread of data includes determining the location of values that divide a set of observations into equal parts such as quartiles. The first quartile (also called the lower quartile) is the data point below which lies the 25 percent of the bottom data. The second quartile (the median) divides the range in the middle and has 50 percent of the data below it. The third quartile (also called the upper quartile) has 75 percent of the data below it and the top 25 percent of the data above it. [4]

For the validation of each prediction model we used 10-fold-cross validation. The data set is broken into 10 sets of equal size. The model is trained with 9 data sets and tested with the remaining tenth data set. This process is repeated 10 times with each of the 10 data sets used exactly once as the validation data. The results of the 10 folds then are averaged to produce the performance measures. We use precision (P), recall (R) for measuring the performance of prediction models. Precision (P) denotes the proportion of correctly predicted Fast bugs: P = TP/(TP + FP). Recall (R) denotes the proportion of true positives of all Fast bugs: R = TP/(TP +FN). [3]

As for the naïve bayes classifier algorithm, it is a simple probabilistic classifier based on applying bayes' theorem with strong (naive) independence assumptions, an advantage of the naive Bayes classifier is that it only requires a small amount of training data to estimate the parameters necessary for classification [2].

III. ANALYSIS

In this section, we prepare the data we will be working with. We gathered data from four software systems, using 17 attributes only from every bug report of the chosen systems, as they are the most commonly used attributes. Applying our hypotheses resulted in a dependent attribute called bugClass.

Firstly, we obtain bug report information from Bugzilla repositories of open source software projects. There are a common set of attributes that we use for each bug report, listed in Table 1. Among the attributes is the fix-time attribute hToLastFix of each bug and it is the time between the opening date till the date of last change of a fixed bug. Also, reporter is the email of the bug reporter. In addition, status is indicating the current state of the bug either resolved or new.

Secondly, we compute our work using naïve bayes algorithm. For each experiment we binned each bug reports into very fast and very slow bugs using the 1st quartile Q1 and 3rd quartile Q3 of hToLastFix respectively as shown in Figure 3.

H1- bugClass =

VeryFast: hToLastFix ≤ Q1

NotVeryFast: hToLastFix > Q1

bugClass=

Fast: hToLastFix ≤ Q2 (Median)

Slow: hToLastFix > Q2 (Median)

H2-bugClass=

NotVerySlow: hToLastFix ≤ Q3

VerySlow: hToLastFix > Q3

Figure3 The classification rules

Table 1. Bug report attributes [3]

Attribute	Short description
Reporter	Email of the bug reporter
Component	Component name of the bug
Assignee	Email of the bug assignee
Priority	Bug priority, e.g. P1,..P5
Severity	Bug severity, e.g. trivial,critical
Platform	Hardware platform, e.g. PC,Mac
OS	Operating system , e.g. windows XP
Resolution	Current resolution, e.g. FIXED
Status	Current status, e.g. NEW, RESOLVED
hToLastFix	Bug fix time(from opened to last fix)
nrActivities	#changes of bug attributes
nrComments	#comments made to a bug report
hOpenedBefore-NextRelease	Hours opened before the next release
monthOpened	Month in which the bug was opened
yearOpened	Year in which the bug was opened
Milestone	Identifier of the target milestone
nrPeopleCC	#people in CC list

IV. EXPERIMENTS RESULTS

We investigated the relationships between the fix-time of bug reports and their attributes with 4 systems taken from the three open source software projects Eclipse, Mozilla, and Gnome. Table 2 shows the number of bug and the observation period for the bug reports. We are classifying bugs with initial bug data using naïve bayes algorithm as shown below. We show the results of our experiments of hypothesis

H1—incoming bug reports can be classified into very fast bugs.

H2— bug reports can be classified into very slow bugs,

In order to prioritize which bugs to start with and which to exclude at the mean time.

Table 2 the number of bugs input to our experiments. [3]

Project	#of bugs	Observation period
Eclipse JDT	10,813	Oct 2001-2007
Mozilla Firefox	8,899	Apr.2001-July 2008
Gnome GStreamer	3,604	Apr.2002- Aug 2008
Gnome evolution	13,459	Jan.1999-july2008

A. Eclipse JDT:

Examining table 3, it shows that using Q1 we will be categorizing the 25th percentile showing the very fast bugs to be fixed that takes less than or equal to 20 hours and the rest of the bugs are not very fast fixed relative to these bugs.

Table 3 Overview of the performance measures obtained by the naïve bayes algorithm using the Q1, Q2, Q3 for EclipseJDT project.

Quartile	target	Prec.	Rec.
Q1 ≤ 20	very fast	0.39	0.20
Q1 > 20	not very fast	0.77	0.90
Q2≤ 122	fast	0.57	0.64
Q2 > 122	slow	0.58	0.51
Q3≤ 675	not very slow	0.78	0.93
Q3> 675	very slow	0.49	0.21

With a recall 0.20 (less than 0.5) which means we miss a lot of bugs that should have been classified as very fast fixed. Rejecting H1 for eclipse JDT using Q1. But on the other hand, categorizing

the not very fast fixed bugs perfectly with recall 0.90. Thus, excluding the slow bugs and clearly focusing on the rest of the bugs in order to start with.

Using Q2 for categorizing the fast and slow bugs with recall 0.64 and 0.51 respectively. Accepting both hypotheses.

Using Q3 we will be categorizing the 75th percentile showing the very slow bugs above Q3 and the rest are the not very slow bugs relative to those defined ones. With a recall 0.49 (less than 0.5) which means we miss a lot of bugs that should have been classified as very slow, we reject H2 for Q3 for eclipse JDT. On the other hand, categorizing the not very slowly fixed bugs perfectly with recall 0.93. Thus, we can classify the not very slowly fixed bugs and the remaining bugs can be classified as very slow ones.

B. *Mozilla Firefox:*

Looking at table 4, it shows that with a recall 0.20 (less than 0.5) which means we miss a lot of bugs that should have been classified as very fast fixed. Thus, we reject H1 for Q1 for Mozilla Firefox. But on the other hand, we categorize the not very fast fixed bugs perfectly with recall 0.91. Thus, excluding the slow bugs and clearly focusing on the rest of the bugs in order to start with.

Table 4 Overview of the performance measures obtained by the naïve bayes algorithm using the Q1, Q2, Q3 for Mozilla Firefox project.

Quartile	target	Prec.	Rec
Q1 ≤ 62	very fast	0.43	0.20
Q1 > 62	not very fast	0.77	0.91
Q2 ≤ 727	fast	0.61	0.65
Q2 >727	slow	0.62	0.58
Q3 ≤ 1846	not very slow	0.81	0.85
Q3>1846	very slow	0.47	0.41

Using Q2 we can classify fast bugs to start with and slow bugs to postpone to be fixed later with recall 0.65 and 0.58 respectively.

Categorizing using the 75th percentile, table 4 shows with a recall 0.41 less than 0.5 which means we miss a lot of bugs that should have been classified as very slowly fixed bugs. Thus, rejecting H2 for Q3. On the other hand, we can categorize the not very slowly fixed bugs perfectly with recall 0.85 and high precision. These should be the bugs that we should start with and the remaining bugs should be avoided.

C. *Gnome GStreamer:*

Table 5 Overview of the performance measures obtained by the naïve bayes algorithm using the Q1, Q2, Q3 for GStreamer project.

Quartile	target	Prec.	Rec.
Q1 ≤ 18	very fast	0.76	0.99
Q1 >18	not very fast	1.00	0.89
Q2 ≤ 128	fast	0.62	0.67
Q2 >128	slow	0.64	0.59
Q3 ≤ 930	not very slow	0.79	0.85
Q3 > 930	very slow	0.41	0.32

Examining table 5, using Q1 we will be categorizing the 25th percentile showing the very fast bugs to be fixed that takes less than or equal to 18 hours and the rest of the bugs are slowly fixed relative to these bugs. Gnome GStreamer outperforms the rest of the projects with a recall 0.99 which means we can easily classify bugs as very fast. In addition, categorizing the slow bugs perfectly with recall 0.89. Thus, we can Accept H1 for classifying very fast bugs so, developers can start with.

Using Q2 will categorize the fast and slow bugs with recall 0.67 and 0.59 respectively.

Using Q3, we get a recall of 0.32 (less than 0.5) which means we miss a lot of bugs that should have been classified as very slowly fixed bugs. On the other hand, we can categorize the not very slowly fixed bugs perfectly with recall 0.85. Thus, we can define the not very slowly fixed bugs and the rest of the bugs are classified as very slow ones.

D. Gnome Evolution:

Table 6 Overview of the performance measures obtained by the naïve bayes algorithm using the Q1, Q2,Q3 for Gnome evolution project .

Quartile	target	Prec.	Rec.
Q1 ≤ 98	very fast	0.44	0.29
Q1 > 98	not very fast	0.79	0.87
Q2 ≤ 701	fast	0. 61	0.65
Q2 > 701	slow	0.63	0.60
Q3 ≤ 3593	not very slow	0.78	0.90
Q3 > 3593	very slow	0. 44	0.25

Looking at table 6, it shows that using Q1 we will be categorizing the 25th percentile showing the very fast bugs to be fixed that takes less than or equal to 98 hours and the rest of the bugs are slowly fixed relative to these bugs. We get a recall of 0.29 (less than 0.5) which means we miss a lot of bugs that should have been classified as very fast. Rejecting H1 for Gnome evolution But, we can categorize the not very fast fixed bugs perfectly with recall 0.87. Thus, excluding the slow bugs and clearly focusing on the rest of the bugs in order to start with.

Using Q2 we will categorize fast and slow bugs perfectly with recall 0.65, 0.60 for fast and slow bugs respectively.

Using Q3 we will be categorizing the 75th percentile showing the very slow bugs 25% above Q3 and the rest are the fast bugs relative to those defined ones. In this case we get with a recall 0.25 which means we miss a lot of bugs that should have been classified as very slow. Rejecting H2 for Gnome evolution. On the other hand, categorizing the not very slow bugs perfectly with recall 0.90. Thus, we can define the not very slowly fixed bugs and the rest of the bugs can be classified as very slowly fixed bugs.

V. RELATED WORK

In the section, we discuss the related work. Hooimeijer and Weimer [5] categorized bug reports into "cheap" and "expensive" using linear regression analysis , on the other hand, we classify bugs into very fast and very slow . Also, Panjer used several different data mining models to predict eclipse bug lifetimes [6]. In Addition, Panjer took into consideration the cc list, dependent bugs, bug dependencies, and comments.

Moreover, Giger et al. [3] has used exhaustive chaid algorithm producing decision tree with median for binning hToLastfix. In our study, we used Q1 and Q3 along with Q2 for binning.

Bhattacharya et al. [7] have used multivariate and univariate regression testing to test the prediction significance of existing models. Weiss et al. predicted person-hours effort spent on fixing that bug using text mining technique to match search reports to new filed bug [8]. Bird et al. proved that there is a systematic bias in bug datasets [9]. This might affect prediction models relying on such biased datasets.

VI. CONCLUSION AND FUTURE WORK

We computed prediction models in a series of experiments with initial bug report data for three active open source projects, our first contribution is that we used quartiles Q1, Q3 for binning instead of using the median only [3] in order to categorize the bugs into very fast bugs so developers can start with and very slow bugs in order to exclude and defer them. In addition, we applied naïve bayes algorithm as recommended by Giger et al. [3] to explore other models with better prediction performance.

On-going and future work is basically concerned with improving the performance of prediction models. We need to examine the priority and severity of the very slow bugs that we are going to exclude to test whether other bugs depend on them to get fixed or not. We also need to extend the input data set. Moreover, we need to take in consideration the post submission data as proven by [3] that it improves the prediction model.

ACKNOWLEDGEMENT

This publication was made possible by NPRP grant # [09-1205-2-470] from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

Also, we would like to thank Dr. Emanuel Giger for all his help and support.

REFERENCES

[1] Bugzilla http://www.bugzilla.org/

[2] I. H. Witten, E. Frank, M. A. Hall, "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann; 3rd edition, 2011.

[3] E. Giger, M. Pinzger, and H. Gall. "Predicting the fix time of bugs", Proc. 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE'10), May 4, 2010, Cape Town, South Africa ,ACM.

[4] R. L. Scheaffer, M. Mulekar, J. T. McClave, *Probability and Statistics for Engineers*, Duxbury Press, 5th edition, 2010.

[5] P. Hooimeijer and W. Weimer, "Modeling bug report quality", In Proc. of the Int'l Conf. on Autom. Softw. Eng., pages 34–43, New York, NY, USA, 2007. ACM.

[6] L. D. Panjer, "Predicting eclipse bug lifetimes". In Proc. Of the Int'l Workshop on Mining Softw. Repositories, page 29,Washington, DC, USA, 2007. IEEE Computer Society

[7] P. Bhattacharya, L. Neamatiu, "bug fix time predicition models:can we do better." http://www.cs.ucr.edu/~pamelab/bugfixtime.pdf

[8] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?", Proc. of the Int'l Workshop on Mining Softw. Repositories, page 1, Washington, DC, USA, 2007. IEEE Computer Society

[9] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced?: bias in bug-fix datasets", Proc. of the Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Foundations of Softw. Eng., pages 121–130, NewYork, NY, USA, 2009. ACM.