Improving Bug Fix-Time Prediction Model by Filtering out Outliers

W. AbdelMoez [1], Mohamed Kholief [2], Fayrouz M. Elsalmy [2]
1. Business Information System Department
2. Department of Computer Science
Arab Academy for Science, Technology and Maritime Transport
P.O. Box 1029, Alexandria, Egypt
walid.abdelmoez@aast.edu , kholief@gmail.com , fayrouz_elsalmy@hotmail.com

***Abstract*— Bug fix time prediction models have been used to predict the fix-time of newly reported bugs in order to help out the developer during the triaging process by prioritizing which bugs to fix first. While constructing these prediction models, we deal with large data sets. It is very likely that these data sets contain outliers that would affect the predictive power of the prediction models. For example, conspicuous bugs are those taking less than a few minutes to get fixed. Also, there are other bugs that take years to get fixed. To improve the quality of the prediction models, a filtering step was proposed to remove these outliers. Our objective is to improve the accuracy of the prediction models by eliminating the effect of these kinds of bugs. Thus in this paper, we examine the distribution of fix-time attribute to identify clearly the potential outliers relative to the data sets. Therefore, we identify several thresholds for filtering out data sets. Also in general, Filtering out using the mild outlier threshold outperforms all other thresholds; bugs were correctly classified into fast which denotes 71% of the fast bugs were classified correctly.**

***Index Terms*— bug fix-time prediction models, filtering outliers, improving prediction models.**

I. INTRODUCTION

The field of Mining Software Repositories (MSR) investigates the available data in software repositories to detect actionable and useful information about software systems and projects. Bug tracking repositories like Bugzilla [1],[2] are used for managing and facilitating the bug resolution process in order to develop and to maintain the software systems effectively. In bug tracking systems, we record different features about the bugs, such as when the bug was reported and which component we found the bug in. In order to coordinate the development effort, two questions need to be addressed: which bug to begin with and how long it will take to fix. Providing software developers with bug categorizing recommendations helps and supports them in cost/benefit analysis. Several efforts have been examined to predict the fix-time of a newly reported bug. In Giger et al. [3], authors used exhaustive chaid algorithm to predict whether the fix time is either slow or fast by the means of decision trees. In AbdelMoez et al. [4], we investigated categorizing the incoming bugs into very fast bugs that the maintainer could start to fix them and, very slow bugs in order to exclude them during the triaging process especially if they do not have a high priority. We classified the bug reports into fast and slow binning bugs using the lower quartile, median, and the upper quartile using naïve-bayes algorithm that was a better predictor than using exhaustive Chaid algorithm. Also, Panjer et al. predicted the fix-time of a bug in Eclipse using logistic regression model providing accuracy up to 34.9% [5]. Bougie et al. used the data from FreeBSD, getting an accuracy of only 19.49% [6]. When considering large datasets, it is common for them to contain irregularities[7]. This is also the case when considering software repositories. In Bird et. al [8], an evidence of systematic bias in bug-fix data sets has been found. In order to improve the prediction model and reduce the effect of irregularities, Lamkanfi et al. [9] used naïve-bayes algorithm and filtering out ½*Q1 getting an accuracy over 60-70%. Similarly in version control systems, Hindle et al. distinguished small commits from large commits [10].

In this paper, we examine the distribution of the fix-time attribute and we use the quartiles to identify the potential outliers. Then, we use them to filter out the data sets. Filtering these outliers can improve the accuracy of the prediction models. We use data from four large data sets of three large open source projects Eclipse, Mozilla and Gnome. We use naïve-bayes classifier with 10-fold cross validation in training and in testing the prediction models. Furthermore, we compare the accuracy before and after filtering out the potential outliers. Then, we evaluate the predictive power of each model using precision, recall and AUC. Our hypothesis is that filtering out the outliers will enhance and improve the bug fix-time prediction models. The main contributions of this paper are:

- First, we propose to filter out the conspicuous bugs by the median of the lower quartile. This shows improvement in the prediction model over the case of filtering them out by half way the lower quartile. Although, we note that both thresholds is not better than the original prediction model.
- Second, we identify several thresholds for bugs taking very long time to get fixed. Results show that the removal of the mild outliers outperforms the rest of the thresholds in almost all of the projects in terms of recall.

The paper is organized as follows. In Section II, we discuss the required background. Section III presents our proposed analysis approach. In Section VI, the experiment and results 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 areas needed in the analysis of the bug fix-time data. As mentioned above, we use quartiles to determine the potential outliers for filtering

out the data sets. We will discuss how to determine the outliers of a dataset. Also, we will hint on using 10-fold-cross validation of prediction models. In the end of this section, we give a brief introduction on Naïve-Bayes algorithm, Wilcoxon test and Friedman test.

A way to describe the spread of data includes determining the location of values that divide an observations set into equal parts such as quartiles. The lower quartile refers to the data point below which lies the 25 percent of the lower data. The median refers to the data point which divides the data into two equal parts. The upper quartile has the top 25 percent of the data above it [11].The difference between third and first quartile is called inter-quartile or IQ. Also, if there is a data value that lies abnormally far from the other values, we call it an outlier. The following thresholds are used for identifying these extreme values in the data sets:

$$\text{Lower inner fence} = Q1 - 1.5*IQ \quad (1)$$

$$\text{Upper inner fence} = Q3 + 1.5*IQ \quad (2)$$

$$\text{Lower outer fence} = Q1 - 3*IQ \quad (3)$$

$$\text{Upper outer fence} = Q3 + 3*IQ \quad (4)$$

If a data point exists beyond an inner fence on either side, we consider it as a mild outlier. Also, if a data point exists beyond an outer fence on either side, we consider it as an extreme outlier [12]. Figure 1 gives an explanation to the quartiles and how to determine the abnormal data from the normal ones by calculating fences as mentioned above.

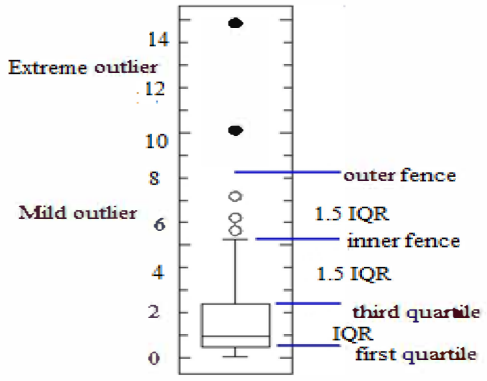


Fig. 1. Identifying Outliers Using Fences

We use 10-fold-cross validation for training and testing the data sets. The data set is broken into 10 data sets of equal size. First, we reserve one tenth of data sets for testing the model. Then, we train the model with the rest of the data set. Finally, we test it with the reserved data set. We repeat this process 10 times with a different testing data set each time. The performance measure of the models is produced by calculating the average of the 10 folds.

Precision (P), recall (R) and receiver operating characteristic curve (AUC) statistic are used for measuring the predictive power of the models. Precision shows the number of true positives divided by the total number of elements that are labeled to be positive. Recall also shows that the number of true positives divided by the total number of the actual positive elements. Moreover, AUC is the area under receiver operating characteristic curve. It is the probability, that, when randomly selecting a fast and a slow bug the model assigns a higher score to the fast bug; as the fast bug is the target class[3]. The higher the value of AUC the higher the classifier's performance in prediction and the higher the value of the recall the better the model at classifying the fast bugs correctly.

The naïve-bayes classifier algorithm is a simple probabilistic classifier. It applies Bayes' theorem assuming naive independence. Small training data set are needed to estimate the parameters necessary for classification. This is considered as an advantage of the naive bayes classifier [13].

Wilcoxon test is a non-parametric statistical hypothesis test. We use this test to compare two related or matched samples to evaluate whether there is difference among the population mean ranks [14]. On the other hand, Friedman test is a test to compare three or more related samples. In Friedman test, there are no assumptions about the underlying distribution of the data. The data is set out in a table comprising n rows by k columns. Then, we rank the data across the rows and we compare the mean rank for each column. [14]

III. STATISTICAL ANALYSIS OF DATA SETS

According to Lamkanfi et al. [9], there are some bugs that take few minutes to get fixed. The way we fix this kind of bugs is different from the manner other reported bugs get fixed. Therefore, they decided to remove these bugs which they called "conspicuous" by setting the threshold of filtering to halfway the lower quartile i.e., the threshold is set to 1/2 *Q1 for each software project separately.

Thus as a first step, we identify the thresholds that we use in this paper. We propose to exclude reports with fix- times below the median of the lower quartile. We examine the data using the following threshold for conspicuous bugs:

- Below halfway the lower quartile
- Below the median of the lower quartile

Then, we compare the predictive power of the model using the half way or using the median to filter out the conspicuous bugs. On the other hand, some other bugs take long time to be fixed, e.g. more than 100 days. As mentioned by lamkanfi et al.[9] after this issue being discussed with Mozilla developers that these bugs were reported with incorrect information making developers lose their attention and thus they do not deal with them. However, if the correct information is provided, the same bugs will be fixed much faster. Therefore, we remove these bugs using several thresholds seeking to improve the prediction models. The filtering thresholds for the bugs with fix-times:

- Above halfway the upper quartile and the maximum
- Above the Median of the upper quartile and the maximum
- Above the upper inner fence, i.e. Mild outliers
- Above the upper outer fence, i.e. Extreme outliers

Then, all the results are compared to the prediction model without filtering called "original model".

We gathered data from four software systems as shown in Table I. First, we get information of the bug reports about open source software projects from Bugzilla repositories. We use 17 attributes only from every bug report of the chosen systems, as

they are the most commonly used attributes; as shown in Table II. 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. In Table III, some basic statistics about the minimum, median and maximum of the fix-times for each of the different cases in our study are shown. Figure 2 shows the box-plots for the bug fix-times attribute obtained from the different software systems data sets expressed in number of hours. Box-plots are used to visualize various statistics of the bug fix-times such as: minimum, lower quartile, median, upper quartile and maximum along with the mild and extreme outliers. Since there are number of software bugs that have very small values and others that have very large value, we present the bug fix-times using a logarithmic scale in the box-plots. Referring to Table III, there are many bugs taking years to be fixed. Other bugs get fixed in seconds. As our data sets are skewed to the right having positive skewness [11] (as shown in Fig. 2.), we do not use lower inner fence calculated using equation (1) and the lower outer fence calculated using equation (3). The lower inner fence and the lower outer fence have negative values and our data is positive as they are time measures.

IV. EXPERIMENT

In this section, we provide the set-up of the experiment. First, we introduce the filtering step for bug reports with different thresholds. Second, we present the results obtained through the experiments. Finally, we compare the accuracy before and after the filtering step.

A. Experiment Setup:

In this paper, several thresholds are used to filter out the conspicuous bugs and the bugs that take very long time to be resolved. The following shows how we compute these thresholds:

- Halfway of the lower quartile :½* Q1
- Median of the lower quartile :median {Min, Q1}
- Halfway of the upper quartile: ½ *[Max-Q3]
- Median of the upper quartile : median {Max, Q3}
- Mild outliers using upper inner fence : Q3 + 1.5*IQ
- Extreme outliers using upper outer fence :Q3 + 3*IQ

Next, we present more details about the steps of setting up of the experiment. First, we want to group the bugs into two categories: Fast and Slow as proposed by Giger et al. [1][3]. The bugClass is the dependent variable as we want to predict this field.

bugClass=
Fast: hToLastFix ≤ Q2 (Median) (5)
Slow: hToLastFix > Q2 (Median)

We categorize a bug report according to the median values shown in Table III. Then, we use the Naive Bayes classifier which is being used in many applications because of its simple principle [15]. During the training phase of the classifier, each independent variable is given a probability value that it belongs to a specific bugClass. When considering a new report, we predict the bugClass using the independent variables in the new report and their respective probabilities. In this case study, we use the Naive Bayes classifier in the WEKA tool [15].

Table I. Selected Software Systems with the Corresponding Number of Bugs and Reporting Periods. [1][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

Table II. 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

Table III. Different Statistics for the data sets.

Project	Minmum (sec)	Median (days)	Maximum (years)
Eclipse JDT	57.6	5.1	5.1
Mozilla Firefox	57.6	15	6.5
Gnome GStreamer	57.6	5.3	4.1
Gnome evolution	57.6	29.2	6.9

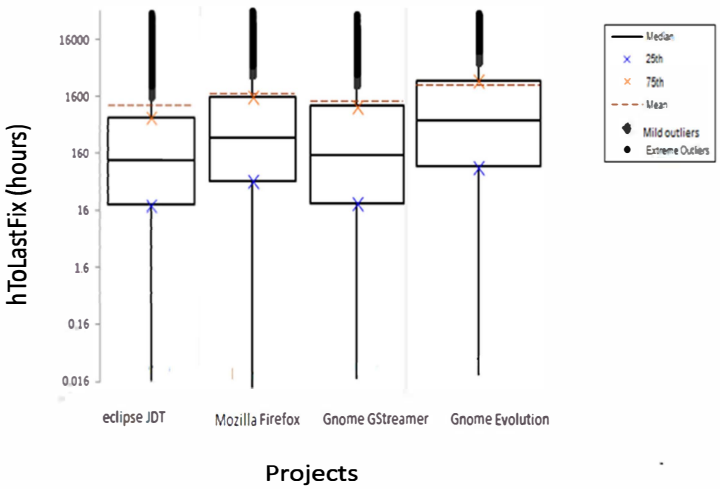


Fig. 2. Box plots of Fix-times Expressed in Hours.

Our experiment has two phases before filtering the outliers and after filtering the outliers. As mentioned before, our predicted models are validated using 10 fold cross validation. The outcome of these two steps are presented and discussed in the next section.

B. Results:

In this section, we present the results obtained from the experiments where we show the accuracy in terms of the AUC, recall and precision before and after we introduce the filtering step.

1) AUC results:

AUC values of each project in our dataset with each threshold are shown in Fig. 3. Filtering out halfway Q3 achieves a median of 0.67 having the same value as the original predictor while the rest of the thresholds exhibit small degradation. Even for the mild threshold, lowest AUC value, is greater than 0.5 which considered to be an effective predictor [16].

In order to conclude if the performance differences between all of the thresholds combined are significant or not, we use Related Samples Friedman Test. The test was significant at α =0.05. This means that there is a statistically significant difference between the mean ranks of the AUC values. Then, we compare the AUC results of each threshold to the original prediction model separately to find out whether the degradation between the AUC values is significant or not using the Wilcoxon test. The test was not significant. Therefore, the decrease in AUC values is not significant. In addition, results show that the median AUC values of approximately 0.64 or higher, which indicate that all classifier models demonstrate adequate performance.

2) Recall

Recall values of each project with different thresholds are shown in Fig. 4. We observe that removal of mild outliers; extreme outliers and removing above the median Q3 are the best thresholds when compared to the original prediction model. Also, filtering by halfway Q3 obtains similarly good results as the original model. Moreover, filtering out halfway Q1 shows lowest performance compared to other thresholds. It has the lowest median for recall and performs the worst on the entire dataset.

In order to measure whether the performance differences between all the thresholds are significant, we use a Related Samples Friedman Test. The results show that test is significant at α= 0.05. This means that there is a statistically significant difference in recall values. Thus, we use Wilcoxon test to compare the recall values of each threshold to the recall of the original prediction model separately to find out whether filtering out outliers help to improve the performance of prediction models. The test was not significant at the level of α=0.05. But the p-values of mild and the extreme outliers are so near to the significance level of α=0.05. If we relax the significance level to α=0.10 (90% confidence level), we can conclude that the improvement caused by the removal of the mild and the extreme outliers are considered statistically significant. Thus, we can argue that the removal of the mild and extreme outliers cause improvement in the recall.

3) Precision:

Figure 5 shows precision values of each prediction model with each threshold for each project. We observe that removal of mild, extreme outliers and removing above the median of Q3 have the best precision values when compared to the original prediction model. Also, filtering out halfway Q3 obtains similarly good results as the original model.

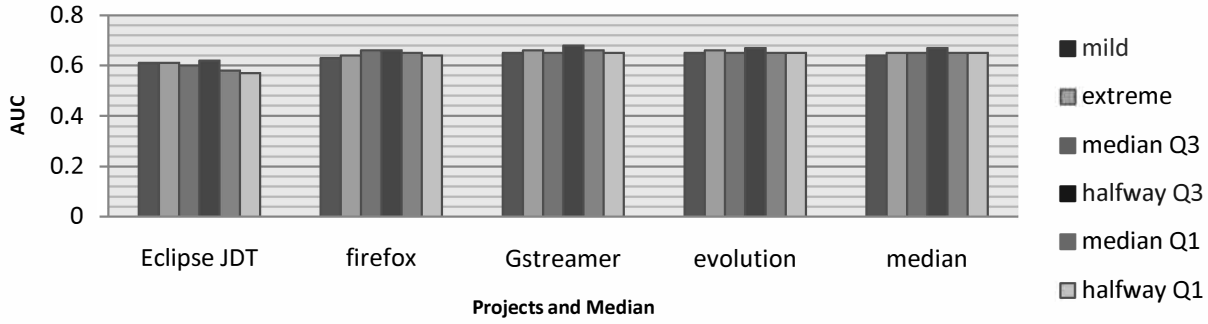
To measure the performance differences between the thresholds in terms of precision, we use Related Samples Friedman Test .The test was significant at α= 0.05. This means that they are statistically significant. Then, we use Wilcoxon test to compare the precision of each threshold to the precision of the original prediction model separately to find out whether filtering out outliers help to improve the performance of prediction models. The test is not significant; therefore we can conclude that improvement in precision caused by filtering out the outliers is not statistically significant.

Finally based on the AUC, recall and precision results, we conclude that removal of mild outliers and extreme outliers outperform the original prediction model. But we need to note

that the improvement is mainly in recall which is significant at α=0.10. This is associated with non-significant improvement in precision and non-significant degradation in AUC.

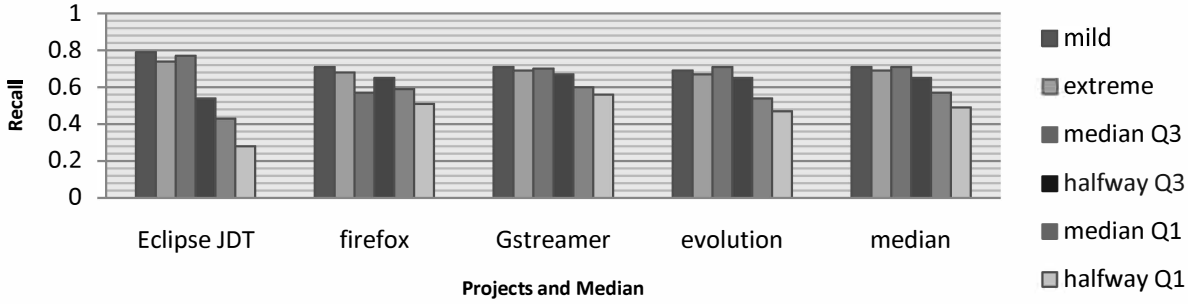
V. RELATED WORK

In the section, we discuss the related work. Lamkanfi et al [9] pointed out that many studies have used data mining algorithms to predict the reported bugs fix-time. In typical open-source projects, the fix-times as reported are heavily skewed. This is mainly because a significant number of reports register fix-times less than a few minutes. They used threshold that is set to ½* Q1 for each software system. They showed improvements in the results outcome when using additional filtering of reported bugs.



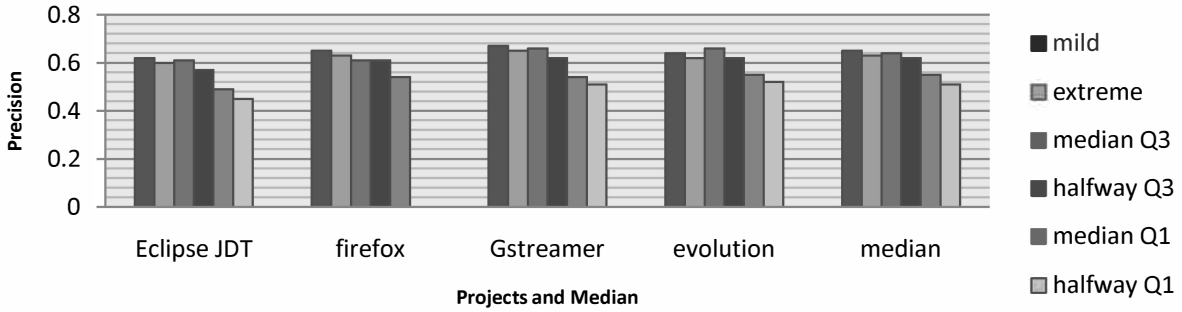
Threshold	mild	extreme	median Q3	halfway Q3	median Q1	halfway Q1	original
median	0.64	0.65	0.65	0.67	0.65	0.65	0.67
p-value- Wilcoxon test	0.066	0.066	0.109	0.317	0.068	0.068	1

Fig. 3 AUC Results for Different Thresholds



Threshold	mild	extreme	median Q3	halfway Q3	median Q1	halfway Q1	original
median	0.71	0.69	0.71	0.65	0.57	0.49	0.65
p-value- Wilcoxon test	0.068	0.068	0.273	0.655	0.593	0.465	1

Fig. 4 Recall Results for Different Thresholds



Threshold	**mild**	**extreme**	**median Q3**	**halfway Q3**	**median Q1**	**halfway Q1**	**original**
median	0.65	0.63	0.64	0.62	0.55	0.51	0.62
p-value- Wilcoxon test	0.713	0.713	0.564	0.317	0.068	0.066	1

Fig.5 Precision results

Hooimeijer and Weimer [17] categorized bug reports into "cheap" and "expensive" using linear regression analysis. Also, Panjer [5] used several different data mining models to predict eclipse bug lifetimes. In Addition, he took into consideration the cc list, dependent bugs, bug dependencies, and comments. Moreover, Giger et al. [3] used exhaustive chaid algorithm producing decision tree with median for binning hToLastfix. Furthermore, Bhattacharya et al. [18] 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 [18][19]. Bird et al. proved that there is a systematic bias in bug datasets[8][8]. This might affect prediction models relying on such biased datasets.

VI. CONCLUSION AND FUTURE WORK

In this research effort, we use data mining techniques in order to predict the bug fix-time of a new bug report. We categorize the bug reports into fast or slow fixed bug in order to help out the developers to decide which bugs to start with. We compute our prediction model using Naïve Bayes algorithm. For each software system, we bin each bug report according to its bug fix time by comparing it with the median value for that project. Thus, we can classify the bug report into fast or slow. Then, we examine the distribution of the bug fix times and we figure out that the removal of the mild and extreme outliers may improve the prediction model.

We perform data mining experiments where we compare the predictions accuracy of the fix-time before and after filtering the outliers from the bug reports. The experiment result shows that the filtering step allowed relatively more accurate prediction models. Again, we need to note that the improvement is mainly in recall. The improvement is statistically significant at 90% confidence level. Also, it is associated with non-significant improvement in precision and non-significant degradation in AUC values.

For future work, we need to analyze other software system projects data to have more evidence of improvement in the prediction models. Also, we need to include other factors in our analysis, e.g.: developer profile, releases, and bugs severity. Also, we need to study the relation of these additional factors to the fix-times.

ACKNOWLEDGMENT

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 and Dr. Ahmed lamkanfi for all their help and support.

REFERENCES

[1] Bugzilla http://www.bugzilla.org/ [last accessed: 10/1/2013]

[2] A. Zeller, J. Krinke, Essential Open Source Tool Set , John Wiley and sons, 2005.

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

[4] W. Abdelmoez, M. Kholeif, F. M. Elsalmy, "Bug fix-time prediction model using naïve bayes classifier", in proc. of the 22nd International Conference on Computer Theory and Applications (ICCTA 2012), pp.167-172, 13-15 October 2012, Alexandria, Egypt.

[5] Panjer, L.D., "Predicting Eclipse Bug Lifetimes," in proc. of 4th International Workshop on Mining Software Repositories, ICSE Workshops MSR '07, pp.29, 20-26 May 2007, Minneapolis, MN, USA.

[6] Bougie, G.; Treude, C.; German, D.M.; Storey, M., "A comparative exploration of FreeBSD bug lifetimes," in proc. of 7th IEEE Working Conference on Mining Software Repositories (MSR), 2010, vol., no., pp.106-109, 2-3 May 2010,Cape Town, South Africa.

[7] J. Maletic and A. Marcus, "Data cleansing: beyond integrity analysis," in proc. Of 5th MIT Conference on Information Quality (IQ 2000), 2000, pp. 200–209, October 2000, Boston, MA., USA.

[8] C. Bird , A. Bachmann , E. Aune , J. Duffy , A. Bernstein , V. Filkov , P. Devanbu, "Fair and balanced?: bias in bug-fix datasets", Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 121– 130, 24-28 August, 2009, Amsterdam, The Netherlands

[9] A. Lamkanfi , S. Demeyer , "Filtering Bug Reports For Fix-Time Analysis" in Proc. of the 16th European Conference on Software Maintenance and Reengineering, pp. 379-384, 27-30 March 2012, Szeged, Hungary

[10] A. Hindle, D.M. German, R. Holt, "What do large commits tell us?: a taxonomical study of large commits", In proc. of the 2008 international working conference on Mining software repositories, pp. 99–108, May 10-11, 2008, Leipzig, Germany.

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

[12] Outliers,http://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm [last accessed:12/2/2013]

[13] N. Fenton, M. Neil, Risk Assessment and Decision Analysis with Bayesian Networks, CRC Press, 1st edition, 2012.

[14] M. Hollander, D.A. Wolfe, Nonparametric statistical methods, Wiley-Interscience, 2nd edition,1999.

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

[16] S. Lessmann, B. Baesens, C. M. Swantje, and Pietsch. "Benchmarking classification models for software defect prediction: A proposed framework and novel findings", IEEE Trans. on Software Engineering, vol.34, no.4, pp.485–496, July-Aug. 2008.

[17] P. Hooimeijer , W. Weimer, "Modeling bug report quality," in Proc. of the twenty-second IEEE/ACM international conference on Automated Software Engineering, pp. 34-43, November 05-09, 2007, Atlanta, Georgia, USA.

[18] P. Bhattacharya , I. Neamtiu, "Bug-fix time prediction models: can we do better?," in Proc. of the 8th working conference on Mining software repositories, pp. 207-210, May 21-22, 2011, Waikiki, Honolulu, HI, USA.

[19] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in proc. of 4th International Workshop on Mining Software Repositories, ICSE Workshops MSR '07, pp.1, 20-26 May 2007, Minneapolis, MN, USA.