Predicting the Fix Time of Bugs

Emanuel Giger
Department of Informatics
University of Zurich
giger@ifi.uzh.ch

Martin Pinzger
Department of Software Technology
Delft University of Technology
m.pinzger@tudelft.nl

Harald Gall
Department of Informatics
University of Zurich
gall@ifi.uzh.ch

ABSTRACT

Two important questions concerning the coordination of development effort are which bugs to fix first and how long it takes to fix them. In this paper we investigate empirically the relationships between bug report attributes and the time to fix. The objective is to compute prediction models that can be used to recommend whether a new bug should and will be fixed fast or will take more time for resolution. We examine in detail if attributes of a bug report can be used to build such a recommender system. We use decision tree analysis to compute and 10-fold cross validation to test prediction models. We explore prediction models in a series of empirical studies with bug report data of six systems of the three open source projects Eclipse, Mozilla, and Gnome. Results show that our models perform significantly better than random classification. For example, fast fixed Eclipse Platform bugs were classified correctly with a precision of 0.654 and a recall of 0.692. We also show that the inclusion of post-submission bug report data of up to one month can further improve prediction models.

1. INTRODUCTION

Several open source projects use issue tracking systems to enable an effective development and maintenance of their software systems. Typically, issue tracking systems collect information about system failures, feature requests, and system improvements. Based on this information and actual project planing, developers select the issues to be fixed. In this paper we investigate prediction models which support developers in the cost/benefit analysis by giving recommendations which bugs should be fixed first. We address the research question whether we can classify incoming bug reports into fast and slowly fixed. In particular, we investigate whether certain attributes of a newly reported bug have an effect on how long it takes to fix the bug and whether prediction models can be improved by including post-submission information within 1 to 30 days after a bug was reported. Intuitively one would expect that some of the attributes, *e.g.*, `priority` have a significant influence of the fix time of a bug. The two hypotheses of our empirical studies are: *H1—Incoming bug reports can be classified into fast and slowly fixed* and *H2—Post-submission data of bug reports improves prediction models*, *e.g.*, number of comments made to a bug.

We investigate these two hypotheses with bug report data of six software systems taken from the three open source projects Eclipse, Mozilla, and Gnome. Decision tree analysis with 10-fold cross validation is used to train and test prediction models. The predictive power of each model is evaluated with precision, recall, and a summary statistic.

2. ANALYSIS

In the first step we obtain bug report information from Bugzilla repositories of open source software projects (see Section 3). For each bug report the set of attributes listed in Table 1 is computed. Some attributes of a bug report, such as the `reporter` and the opening date, are entered once during the initial submission and remain constant. Other attributes, such as `milestone` and `status`, are changed or entered later on in the bug treating process. We highlight attributes that remain constant over time in Table 1 by an *I* and attributes that can change by a *C*. The change history of bug reports is stored in bug activities. We then use the change history of bug reports to compute the measures marked with *C* at specific points in time. In addition to the initial values we obtain the attribute values at 24 hours (1 day), 72 hours (3

Table 1: Constant (I) and changing (C) bug report attributes.

Attribute	Short Description
`monthOpened`, *I*	month in which the bug was opened
`yearOpened`, *I*	year in which the bug was opened
`platform`, *C*	hardware plaform, *e.g.*, PC, Mac
`os`, *C*	operating system, *e.g.*, Windows XP
`reporter`, *I*	email of the bug reporter
`assignee`, *C*	email of the bug assignee
`milestone`, *C*	identifier of the target milestone
`nrPeopleCC`, *C*	#people in CC list
`priority`, *C*	bug priority, *e.g.*, P1, ..., P5
`severity`, *C*	bug severity, *e.g.*, trivial, critical
`hOpenedBefore-NextRelease`, *I*	hours opened before the next release
`resolution`, *C*	current resolution, *e.g.*, `FIXED`
`status`, *C*	current status, *e.g.*, `NEW`, `RESOLVED`
`hToLastFix`, *I*	bug fix-time (from opened to last fix)
`nrActivities`, *C*	#changes of bug attributes
`nrComments`, *C*	#comments made to a bug report

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
RSSE '10, May 4 2010, Cape Town, South Africa
Copyright 2010 ACM 978-1-60558-974-9/10/05 ...$10.00.

days), 168 hours (1 week), 336 hours (2 weeks), and 720 hours (~1 month) after a bug report was opened. `nrActivities` simply refers to the number of these changes up to a given point in time. `nrComments` is similar but counts the number of comments entered by Bugzilla users up to the given point in time. The fix-time `hToLastFix` of each bug report is measured by the time between the opening date and the date of the last change of the bug resolution to `FIXED`.

In a second step we computed decision trees using Exhaustive CHAID algorithm [6]. For each experiment we binned bug reports into `Fast` and `Slow` using the median of `hToLastFix`:

$$bugClass = \begin{cases} Fast & : \ hToLastFix <= median \\ Slow & : \ hToLastFix > median \end{cases}$$

`bugClass` is the dependent variable with `Fast` selected as target category. The remaining bug measures are used as independent variables in all of our experiments. Because both bins are of equal size, the prior probability for each experiment is 0.5 which corresponds to random classification. We used the default settings of 100 for the minimum number of cases for parent nodes and 50 for the minimum number of cases in leaf nodes. The tree depth was set to 3 levels.

For the validation of each prediction model we used 10-fold-cross validation [8]. 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), and the area under the receiver operating characteristic curve (AUC) statistic 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: $R = TP/(TP + FN)$. AUC is the area under receiver operating characteristic curve. It can be interpreted as the probability, that, when randomly selecting a positive and a negative example the model assigns a higher score to the positive example [4]. In our case the positive example is a bug classified `Fast`.

3. EXPERIMENTS

We investigated the relationships between the fix-time of bug reports and their attributes with six (sub-)systems taken from the three open source software projects Eclipse, Mozilla, and Gnome. Table 2 lists the number of bugs input to our experiments.

Table 2: Number of bugs and dates of first and last filed bug reports of subject systems.

Project	#Bugs	Observation Period
Eclipse JDT	10,813	Oct. 2001 – Oct. 2007
Eclipse Platform	11,492	Oct. 2001 – Aug. 2007
Mozilla Core	27,392	Mar. 1997 – June 2008
Mozilla Firefox	8,899	Apr. 2001 – July 2008
Gnome GStreamer	3,604	April 2002 – Aug. 2008
Gnome Evolution	13,459	Jan. 1999 – July 2008

3.1 Classifying Bugs with Initial Bug Data

In this section we present the results of our investigation of hypothesis *H1—incoming bug reports can be classified into fast and slowly fixed*. Table 3 gives an overview of the performance measures obtained by the decision tree analysis. We used `Fast` as target variable for our calculations.

Table 3: Performance measures of prediction models computed with initial attribute values.

Project	Median	Prec.	Rec.	AUC
Eclipse JDT	122	0.635	0.485	0.649
Eclipse Platform	258	0.654	0.692	0.743
Mozilla Core	727	0.639	0.641	0.701
Mozilla Firefox	359	0.608	0.732	0.701
Gnome GStreamer	128	0.646	0.694	0.724
Gnome Evolution	701	0.628	0.695	0.694

Eclipse.

Looking at Table 3 we see that the decision tree model obtained with Eclipse Platform bug reports outperforms the Eclipse JDT model. The most important attribute in the Eclipse Platform model is `monthOpened`. An investigation of the values, however, yielded no clear trend that bug reports are treated differently during the year. The second attribute attached to the tree is `assignee`.The model performance is significantly higher than random classification which lets us accept hypothesis H1 for Eclipse Platform.

With a low recall value of 0.485 the Eclipse JDT model strikes out. A recall value lower than 0.5 indicates that the model misses more than half of `Fast` bug reports. Furthermore, the Eclipse JDT model has the lowest AUC value of all examined projects. The top most attribute of the Eclipse JDT decision tree is `assignee`. The overall structure of the tree affirms the moderate performance of the model. Most of the nodes in the decision tree show low performance to distinguish between fast and slowly fixed bugs. We reject hypothesis H1 for Eclipse JDT.

Mozilla.

Decision tree models computed with bug reports of the two Mozilla projects show similar performance. The first attribute considered in the decision tree of the Mozilla Core project is `yearOpened`. Bug reports opened after the year 2003 were more likely to get fixed fast with a probability of 0.632. In contrast, bug reports opened before 2001 tend to be classified `Slow` with a probability of 0.639. Bug reports opened between 2001 and 2003 cannot be distinguished sufficiently by `yearOpened`. Additionally, the decision tree model contains the `component` of a bug as well as information about the `assignee`, the operating system (`os`), and `monthOpened`. Improvements over random classification are significant and we accept hypothesis H1 for Mozilla Core.

In contrast to Mozilla Core, the Firefox model contains `component` as the most significant predictor. There is one node predicting perfectly, however, it only covers 0.9% of bug reports. The second most important attribute is the `assignee`, and in contrast to the Mozilla Core model, the `yearOpened` attribute of Firefox bug reports is of only minor relevance. Precision, recall, and AUC values let us accept hypothesis H1 for Mozilla Firefox.

Gnome.

The prediction models of both Gnome projects improve random classification. The top most attribute of the Gnome GStreamer decision tree is `yearOpened`. Similar to Mozilla

Core older bug reports (*i.e.*, opened before 2005) were likely to take more time to fix than recently reported bugs. The affected component is the second most significant predictor. An investigation of corresponding tree nodes showed that bug reports which affected components related to the plugin architecture of Gnome GStreamer tend to be fixed faster. In particular recent bug reports followed this trend. As in our previous experiments prediction models were improved by including the attributes reporter and assignee. The values for precision, recall, and AUC let us accept hypothesis H1 for Gnome GStreamer.

The decision tree model of Gnome Evolution bug reports contains assignee as first attribute. The attributes on the second level of the tree are hOpenedBeforeNextRelease, reporter, yearOpened, and severity. An investigation of the decision tree did not show any patterns or tendencies, that enable a straight forward classification of bug reports into Slow and Fast. Concerning precision, recall, and AUC the model performs significantly better than random classification. We accept hypothesis H1 for Gnome Evolution.

In summary, decision tree analysis with the initial bug attributes obtains prediction models that for five out of six systems perform 10 to 20% better than random classification. This is a sufficient indicator that we can compute prediction models to classify incoming bug reports into Fast and Slow and we accept hypothesis H1.

3.2 Classifying Bugs with Post-Submission Data

This section presents the results of the evaluation of hypothesis *H2—post-submission data of bug reports improves prediction models*. For each bug report we obtained post-submission data at different points in time, namely 1 day, 3 days, 1 week, 2 weeks, and 1 month after the creation date of the bug report. For each observation period we computed decision tree models which we validated with 10-fold cross validation. The following paragraphs present and discuss the results of experiments and performance measures of prediction models.

Eclipse.

Table 4 lists the median fix-time of bugs and the results of decision tree analysis with bug reports of the Eclipse JDT project.

Table 4: Median fix-time and performance measures of Eclipse JDT prediction models.

Days	Median	#Bugs	Prec.	Rec.	AUC
0	122	10,813	0.635	0.485	0.649
1	296	7,732	0.710	0.577	0.742
3	491	6,277	0.693	0.659	0.767
7	865	4,767	0.750	0.606	0.785
14	1,345	3,653	0.775	0.661	0.823
30	2,094	2,615	0.885	0.554	0.806

The inclusion of post-submission information improved the performance of prediction models as indicated by increasing precision, recall, and AUC. In contrast to the initial decision tree, the models built with post submission data obtained milestone as the top most predictor. New bug reports rarely have a milestone specified which, in the case of Eclipse JDT, are 36 out of 10,813 bug reports. Within one week the ratio of pending bugs with milestones increased to 37% and afterwards remained constant. The inclusion of milestone led to improved performance of prediction models for the Eclipse JDT project. In addition to milestone, the assignee, the reporter, monthOpened, and yearOpened represent significant predictors in computed decision tree models. The best performing model takes into account 14 days of post-submission data. Precision, recall, and AUC values of this model are higher as the corresponding values of the initial model. This lets us accept hypothesis H2 for Eclipse JDT.

Table 5 lists the performance measures for the Eclipse Platform bugs. Experiments showed similar results as before with Eclipse JDT. On average, bugs in the Eclipse Platform project tend to take longer to fix than in the Eclipse JDT project. This is indicated by a higher median fix-time for the different observation periods.

Table 5: Median fix-time and performance measures of Eclipse Platform prediction models.

Days	Median	#Bugs	Prec.	Rec.	AUC
0	258	11,492	0.654	0.692	0.743
1	560	9,003	0.682	0.586	0.734
3	840	7,803	0.691	0.631	0.749
7	1,309	6,457	0.691	0.587	0.738
14	1,912	5,307	0.743	0.669	0.798
30	2,908	4,135	0.748	0.617	0.788

The inclusion of post-submission data of Eclipse Platform bug reports only sightly improved prediction models. As in the decision tree computed with Eclipse JDT bug reports, the milestone attribute was selected as the first attribute in the tree. Also in the Platform data, milestones are added in the post-submission phase of bug reports. After one day, milestones were added to 27% of pending bugs. This ratio remained constant for the later observation points. Most of the undecidable bugs do not have any milestone specified. The monthOpend, reporter, and assignee are the other significant predictors contained by decision tree models. The model with 14 days of post-submission data performed best. Improvements over the initial model led to the acceptance of hypothesis H2 for Eclipse Platform.

Mozilla.

The results of the decision tree analysis with bug reports of the Mozilla Core project are depicted in Table 6. The median bug fix-time indicate longer fix times for Mozilla Core than for Eclipse bugs on average.

Table 6: Median fix-time and performance measures of Mozilla Core prediction models.

Days	Median	#Bugs	Prec.	Rec.	AUC
0	727	11,377	0.639	0.641	0.701
1	935	10,424	0.708	0.667	0.773
3	1,179	9,524	0.727	0.630	0.770
7	1,617	8,347	0.712	0.697	0.777
14	2,201	7,142	0.757	0.671	0.803
30	3,257	5,716	0.688	0.708	0.746

Mozilla Core models contained priority, milestone, assignee, and reporter as significant predictors. priority is the first attribute in decision tree models computed with 3 and 7 days of post-submission data. Bug reports with low priority take longer to fix than bugs with higher priority. For example, in the 3-days model 80.7% of 1,255 bug reports with priority P1 were fixed fast. milestone is the most significant predictor in the other models that consider post-submission data. In Mozilla Core few (1.6%) milestones were entered when the bug was reported. This ratio changed to 30% within one day whereas most of the reports were assigned to the "moz" milestone. The ratio steadily increased up to 47% within 30 days after bug report submission. In extension to Eclipse JDT and Platform, the models computed with Mozilla Core bug reports contained also `severity`, the affected `component`, `nrComments`, and `nrActivities`. Prediction models with post-submission data show improved performance, hence, we accept hypothesis H2 for Mozilla Core. The median fix-time and performance measures of models computed with Mozilla Firefox bugs are listed in Table 7. The median fix-time indicates faster fixes of Mozilla Firefox bugs than Mozilla Core bugs.

Table 7: Median fix-time and performance measures of Mozilla Firefox prediction models.

Days	Median	#Bugs	Prec.	Rec.	AUC
0	359	8899	0.609	0.732	0.701
1	587	7478	0.728	0.584	0.748
3	801	6539	0.697	0.633	0.742
7	1176	5485	0.729	0.610	0.759
14	1778	4553	0.680	0.683	0.757
30	2784	3440	0.751	0.748	0.834

The best model was computed with post-submission data of up to 30 days. This decision tree model has a precision of 0.751, a recall of 0.748, and an AUC of 0.834. While in previous prediction models `milestone` or `priority` were selected as the most significant predictors, `nrActivities` and `yearOpened` were selected in Mozilla Firefox models. In the models computed with 1, 3, and 7 days of post-submission data we observed, that bugs with zero or one activity were fixed slower than bugs with more than 7 activities. The ratio of bug reports with specified milestones follows a similar trend as in previous case studies. Surprisingly, the model with the best performance (30 days) does not contain the `milestone` attribute. In this model, `yearOpened` is the most significant predictor. In particular, bugs that were reported before the year 2003 took longer to fix on average than bugs reported after the year 2006. The `reporter` and `assignee` were the other bug attributes contained by this decision tree. The good performance of the last model (30-days) lets us accept the hypothesis H2 for Mozilla Firefox.

Gnome.

Table 8 lists the measures of the prediction models computed with the Gnome GStreamer bug reports. Similar to bug reports of the two Eclipse projects many reports in Gnome GStreamer have a short fix-time on average as indicated by lower median fix-time.

Table 8: Median fix-time and performance measures of Gnome GStreamer prediction models.

Days	Median	#Bugs	Prec.	Rec.	AUC
0	128	3604	0.646	0.694	0.724
1	406	2553	0.581	0.810	0.666
3	708	2052	0.606	0.704	0.667
7	1084	1650	0.613	0.652	0.669
14	1517	1351	0.658	0.561	0.680
30	2268	1018	0.538	0.811	0.586

In contrast to previous experiments, the performance of models computed for Gnome GStreamer decreases with the inclusion of post-submission information. While the AUC value of the initial model is 0.724 the AUC value of the last model is only 0.586. One big difference is that in Gnome GStreamer the `milestone` attribute is specified for only few bug reports, hence, was not included into prediction models. Although, milestones were initially specified for 9% of bug reports, this ratio increased to only 18% within 30 days which is lower than the ratio in the Eclipse or Mozilla projects. In the models with post-submission data, the `assignee` is the most significant predictor followed by the `reporter` and `nrComments`. Also with post-submission data we could not obtain reasonable prediction models, hence, we reject hypothesis H2 for the Gnome GStreamer.

The next series of experiments was with bug reports of the Gnome Evolution project. The results of the decision tree analysis are depicted by Table 9. Bugs of this system tend to take longer to fix on average than in the other subject systems.

Table 9: Median fix-time and performance measures of Gnome Evolution prediction models.

Days	Median	#Bugs	Prec.	Rec.	AUC
0	701	13459	0.628	0.695	0.694
1	1136	11548	0.649	0.659	0.727
3	1476	10496	0.693	0.611	0.746
7	1962	9335	0.636	0.798	0.752
14	2566	8228	0.665	0.760	0.766
30	3625	6695	0.690	0.682	0.771

Compared to Gnome GStreamer the models computed with the Gnome Evolution bug reports show better performance regarding precision, recall, and AUC values. The performance of the decision tree models increases when including post-submission information. Similar to the decision trees computed with Eclipse and Mozilla bug reports `milestone` is the most significant predictor followed by `assignee`. Milestones were added for 21% of the bugs within one day. This ratio increased to 31% within 30 days. Slow bug reports are indicated by milestones, such as "Later", "Future", or "reschedule" while fast bug reports got mainly concrete release numbers. Bug reports with no milestone are basically undecidable. Other significant predictor variables which appeared in the various Gnome Evolution models are the `reporter`, `yearOpened`, and `monthOpened`. Furthermore, `severity` and `hOpenedBeforeNextRelease` are significant. The good performance of the prediction models with 7 and 14 days of post-submission data lets us accept hypothesis H2 for Gnome Evolution.

In summary, the inclusion of post-submission data led to improved predictive power of models in all systems but Gnome GStreamer. We therefore accept hypothesis H2.

4. RELATED WORK

Hooimeijer and Weimer [5] used linear regression analysis on bug report data to predict whether a bug report is triaged within a given amount of time. Similar to our approach they take into account post-submission data and investigate how much of this data is needed to yield adequate predictive power. While they focus on reducing the bug triage time which they denote as time needed to inspect, understand, and making the initial decision regarding how to address the report, we concentrate on the fix-time of bugs. Furthermore, they aim at finding an optimal cut-off value to classify bug reports into "cheap" and "expensive" while we use fixed cut-off values for `Fast` and `Slow`. Additionally, we use decision tree analysis instead of linear regression analysis.

Another important basis for our work was done by Panjer in [10]. He used several different data mining models to predict eclipse bug lifetimes. We extend his work by looking at more systems. Furthermore, while he counted the cc list, dependent bugs, bug dependencies, and comments we take into account that other attributes, *e.g.*, `assignee` might change as well. We rather create different profiles representing the state of a bug at a certain point of its lifetime.

An approach to assist in bug triage is presented by Anvik *et al.* in [1]. They give suggestions to which developer a new bug report should be assigned. To find suitable developers among all possible candidates they apply machine learning techniques to open bug repositories. It would be interesting to see whether vector support machines instead of decision trees can improve prediction in our sense.
Wang *et al.* recognize the mentioned problem of duplicates and its possible drawbacks on bug triage [11].

Kim and Whitehead argue that the time needed to fix a bug is a significant factor when measuring the quality of a software system [7]. Our approach is complementary, in that it provides a prediction model for estimating whether a bug will be fixed fast or take more time for resolution.

Given a new bug report Weiss *et al.* present a method to predict the effort, *i.e.*, the person-hours spent on fixing that bug [12]. They apply text mining technique to search reports that match a new filed bug. They use effort measures from past bug reports as a predictor. We also use existing data from recoded bug reports to compute a prediction model but we remain limited to non-textual features of a bug report.

Bettenburg *et al.* investigated which elements developers rely on when fixing a bug [2]. Similar to our approach they claim that the information given in bug reports has an impact on the fix time.

Lessmann *et al.* compare different classification models for software defect prediction using AUC as benchmark [9]. We use similar analysis techniques and performance evaluation criteria but instead of failure-proneness aim at providing models to predict the fix time of bugs.

Recently Bird *et al.* have found evidence the there is a systematic bias in bug datasets [3]. This might effect prediction models relying on such biased datasets.

5. CONCLUSIONS & FUTURE WORK

We computed prediction models in a series of experiments with initial bug report data as well as post-submission information from three active open source projects. Summarized, the results of our experiments are: Between 60% and 70% of incoming bug reports can be correctly classified into fast and slowly fixed. `assignee`, `reporter`, and `monthOpened` are the attributes that have the strongest influence on the fix-time of bugs. Post-submission data of bug reports improves the performance of prediction models by 5% to 10%. The best-performing prediction models were obtained with 14-days or 30-days of post-submission data. The addition of concrete `milestone` information was the main factor for the performance improvements (see Section 3.2). Decision tree models with initial and post-submission bug report data showed adequate performance when compared to random classification. However, the applicability of these models to develop fully automated recommender systems is questionable. We can think of recommender tools in a way that they provide valuable input to developers and aid them in deciding which bugs to address first—though, their decision can not be solely based on the output of the models. Our models could also be useful to new developers or bug reporters as they give an insight in how bugs are prioritized in a software project.

On-going and future work is basically concerned with improving the performance of prediction models. For this we plan to extend the input data set and investigate other algorithms to compute prediction models. For example, detailed change information of bug report attributes, data about the affected components and text analysis will be tested. Furthermore, we plan to evaluate whether Random Forests and Naive Bayes algorithms can improve prediction models.

6. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proc. of the Int'l Conf. on Softw. Eng.*, pages 361–370, New York, NY, USA, 2006. ACM.

[2] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proc. of the Int'l Symp. on Foundations of Softw. Eng.*, pages 308–318, 2008.

[3] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced?: bias in bug-fix datasets. In *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, New York, NY, USA, 2009. ACM.

[4] D. M. Green and J. A. Swets. *Signal Detection Theory and Psychophysics*. John Wiley & Sons, Inc., New York NY, 1966.

[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] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of Applied Statistics*, 29(2):119–127, 1980.

[7] S. Kim and J. E. James Whitehead. How long did it take to fix bugs? In *Proc. of the Int'l Workshop on Mining Softw. Repositories*, pages 173–174, New York, NY, USA, 2006. ACM.

[8] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of the Int'l Joint Conf. on Artificial Intelligence*, pages 1137–1143. Morgan Kaufmann, 1995.

[9] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Trans. on Softw. Eng.*, 34(4):485–496, 2008.

[10] 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.

[11] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proc. of the Int'l Conf. on Softw. Eng.*, pages 461–470, New York, NY, USA, 2008. ACM.

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