Automated Classification of Faults in Programs using Machine Learning Techniques

Javed Ferzund and **Syed Nadeem Ahsan** and **Franz Wotawa**

Abstract. It is always argued that software complexity metrics are an important quality indicator of the developed software. In this paper, we focus on the question whether software complexity metrics can be used for predicting bugs in software. In contrast to other research in this area we are interested in obtaining a predictor for classifying source code files into the three categories: low, medium, and high. For this purpose, we apply machine learning techniques. We present results obtained when analysing the different versions of the open source software Firefox. One outcome of the study is that the obtained predictor is highly accurate.

1 Introduction

Software testing requires a lot of effort and resources. To efficiently utilize the available resources for testing , it is better to schedule these resources to those parts of the source code where the likelyhood of bugs is greater. A lot of work have been done for predicting bugs in several different ways. Most of the studies indicate a relationship between the obtained metrics and the number of bugs like [9] and [14]. The research relies on software configuration management systems and bug databases hold important information related to bugs and changes made to files. Other approaches are mainly used to predict the number of bugs [7].

In order to predict the number of bugs or to provide a predictor with regard to a classification schema there are two approaches possible. The first approach uses statistical methods like multiple linear regression, logistic regression, and principal components analysis[12]. Linear regression can be successfully used if the dependent variables change linear with the independent variables. As most of the metrics normally correlate with each other , there is a strong need to overcome the multicolinearity problem. Principal component analysis is used in this respect to reduce the multicolinearity effect. Logistic regression can be used for binary classifications.

The second approach relies on machine learning techniques like decision tree induction, support vector machine, artificial neural networks, k-nearest neighbors to mention some of them. Machine learning techniques have the abilty to learn from past data and these techniques can be employed in a variety of complex situations (see [18]).

In this paper, we make use of machine learning techniques in order to provide a predictor to classify source files accordingly to a pre-defined classification schema. In the paper we compare two classification schemes. The first schema allows to classify files as being buggy or not. The second schema is called bug level classification where files are classified as low buggy, medium buggy, or highly buggy respectively.

- *isBuggy*: If a file contains no bugs, its *isBuggy* value is set to “no”, otherwise the value is set to “yes”.
- *bugLevel*: We define three bug levels based on the number of bugs in each file as follows:

Number of Bugs	Bug Level
0-10	low
10-30	medium
30 and above	high

In our work, we use different machine learning techniques for automatically obtaining a classifier from one version of a program. In order to judge the quality of the obtained classifier, we use other versions of the same program in order to test the classifier. In particular, we used Firefox 0.8 for extracting the classifier, and versions 1.0, 1.5, and 2.0 to test them. We obtained the used bug information from the concurrent version system and bugzilla. We calculate the metrics for each file using a self-developed software. The obtained results for the *bugLevel* classifier is very promising. In more than 90 % of the cases the automatically obtained classifier returns the correct results. This is not the case for the *isBuggy* classifier, which has an average accuracy of only about 50 %.

The paper is organized as follows. We first discuss related research. Afterwards, we introduce our approach and present the obtained empirical results. Finally, we conclude the paper.

2 Related work

A lot of research has been carried out regarding complexity metrics and prediction of bugs. Chidamber and Kimerer [6] initially proposed the object-oriented metrics. Gyimothy et al. [9] validated the object-oriented metrics for fault prediction in open source software. In this paper the authors used logistic regression and machine learning techniques to identify faulty classes in mozilla. Ostrand et al. [14] used the source code of the file in current release, and the fault and modification history of the previous releases to predict the expected number of faults in the source files of the next release. The top 20% of the files with highest predicted number of bugs contained more than 70% of the defects detected. Zhang [19] showed the application of machine learning techniques in tackling software engineering problems.He suggested that machine learning algorithms can be used to build tools for software development and maintenance tasks as well as can be incorporated into software products to make them adaptive and self-configuring.

Porter and selby [15] used classification trees based on metrics from previous releases to identify components having high-risk properties. The authors developed a method of automatically generating measurement-based models of high-risk components.They classified modules according to error proneness, development cost,rework effort and containing a specified class of error.Fischer et al. [8] com-

bined the data from version control systems and bug tracking systems and provided a way for mapping bugs to source code locations.

Brun and Ernst [5] used machine learning techniques to model program properties that result in errors.The authors applied these models to classify and rank properties of user written programs that may result in errors.The technique proposed by them could select a subset of properties which were more error prone.They used support vector machines and decision trees for their experiments. Shepperd and Kadoda [16] simulated data sets for comparing four prediction techniques including regression, rule induction, nearest neighbor (a form of case-based reasoning), and neural nets. They concluded that the results of the prediction technique vary with the characteristics of the data set being used. They suggested that it is good to ask, find best prediction system for a particular context rather than finding universally best prediction system.

Zimmerman et al. [20] applied data mining techniques to version histories to find association rules, which can be used to suggest or predict further changes based on a given change. After an initial change, their proposed prototype could correctly predict 26% of the further files to be changed, and 15% of the functions or variables to be changed. The topmost three suggestions contained a correct location with a likelihood of 64%. Aljahdali et al. [3] used feed-forward neural network to predict the faults in a program during the initial test/debug process.they also compared the results between regression models and neural networks.

Neumann [13] used principal component analysis and artificial neural networks for software risk categorisation. He provided a technique with the capability to discriminate data sets that include disproportionately large number of high-risk software modules. Fenton and Neil [7] provided a critical review of the defect prediction models based on software metrics. They discussed the weaknesses of the models proposed previously and identified causes for these shortcomings. Kim et al. [10] applied machine learning techniques for classifiying software changes as clean or buggy. They have trained the classifier on features extracted from the revision history of a software project. The classifier trained by them, was able to classify changes as clean or buggy with a 78% accuracy, which is in contradiction to the results we obtained.

Venkata et al. [17] applied different machine learning techniques to develop a predictor model. They used models on real time software defect data and concluded that a combination of 1R and Instance-Based learning along with consistency based subset evolution techniques provides better results as compared to other techniques Boetticher [4] performed an experiment to show how biased data distribution impacts the results and presented two nearest neighbor sampling approaches.The author also showed how these approaches may be incorporated into data mining process.Koru et al. [11] combined static software measure with defect data at class level and applied different machine learning techniques to develop bug predictor model.

3 The classification approach

In this section, we describe the tasks that had been carried out in order to obtain the empirical results. The motivation is to provide background information. We start with a discussion of how to obtain the data collection. Afterwards, we introduce the concepts behind the metrics calculation. Finally, we briefly describe the used machine learning techniques.Figure 1 represents a schematic diagram of the classification approach.

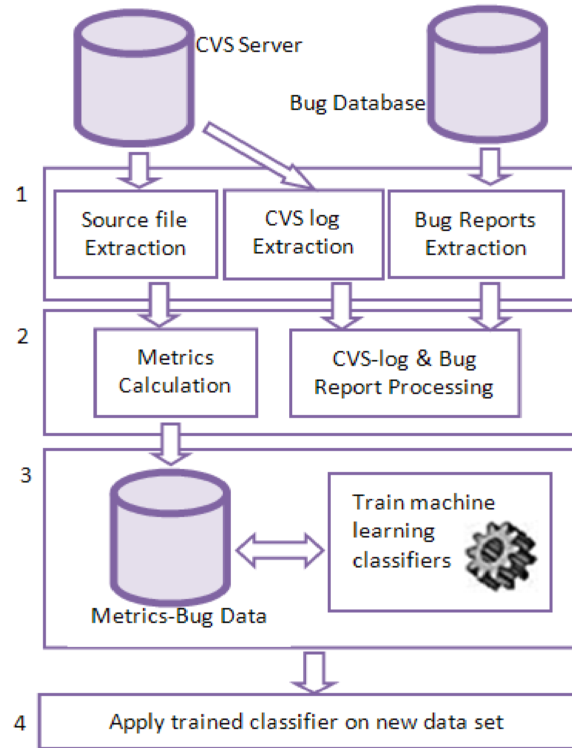

Figure 1. Classification Approach

Collecting data: We downloaded Firefox source code (Release 0.8, 1.0, 1.5 and 2.0) on a local drive using the ftp server. We obtained the bug information from Bugzilla[1]. We also processed the CVS log output to find the revisions of files in which bugs were fixed. A database was designed to hold the information regarding bugs and metrics. The bugs reported in Bugzilla as well as the CVS comments were used to map bugs to individual revisions of files. We calculated for each file the number of bugs that occurred between two consecutive releases of Firefox.We processed the following number of cpp,header and c files in ech release of Firefox:

File Type	Firefox 0.8	Firefox 1.0	Firefox 1.5	Firefox 2.0
cpp	3913	4017	4216	4276
header	2669	2731	2821	2879
c	1389	1387	1522	1527

Calculation of metrics data: We used our own software to calculate the source code metrics.We calculated individual metrics for functions and classes. These metrics were then aggregated on files getting total and maximum values of each metrics for each file.The function metrics include the following:

Metrics	Description
CC	Cyclomatic Complexity
CD	Control Density
PC	Parameter Count
RPC	Return Point Count
LVC	Local Variable Count
ND	Nesting Depth
FLOC	Function Lines Of Code

The class metrics include the following:

Metrics	Description
NOP	Number Of Parents
NOC	Number Of Children
MFC	Member Function Count
DMC	Data Member Count
ID	Inheritance Depth
CBO	Coupling Between Objects

File metrics include the following along with the aggregated metrics for functions and classes:

Metrics	Description
NOF	Number Of Functions
NOCS	Number Of ClaSses
NOIF	Number Of Included Files
LOC	Lines Of Code

Application of machine learning techniques: We used WEKA (a Machine Learning tool developed in JAVA) [2] for obtaining a classificator for files based on the number of bugs and the obtained complextiy metrics. Seven different machine learning approaches are used to obtain different classifiers. All of them were trained on metrics and bug data of Firefox Release 0.8 according to both the *isBuggy* and the *bugLevel* criteria. The obtained models were afterwards tested in order to classify the files of later Firefox releases.

4 Results

We applied the models obtained from Firefox Version 0.8 to the data of Firefox releases 1.0, 1.5 and 2.0. We discuss the obtained results for the different machine learning techniques in this section. For the machine learning techniques decision tree (D Tree), k-nearest neighbors (KNN), naive Bayes (NB), support vector machine (SVM), radial basis function network (RBF), decision table (D Table), and artificial neural networks (ANN) we obtained the number of correctly classified instances (CCI), the Kappa statistics (KS), the mean absolute error (MAE), and the root mean squared erros (RMSE).KS measures the agreement between two observers.KS values near 1 indicate good agreement and negative values indicate that the two observers agreed less than would be expected just by chance.MAE measures the weighted average of the absolute errors.The RMSE measures the average magnitude of the error. It gives a relatively high weight to large errors. MAE and RMSE can range from 0 to ∞.Lower values indicate that the prediction technique is more accurate.

Using the *isBuggy* criteria we correctly classified 75% of the cpp files except Firefox 1.0, for which the correct classification was 42%. Using the *bugLevel* criteria, we correctly classified more than 90% of the cpp files. Table1 shows that for Firefox 1.0 KS values are negative which indicate that there is no agreement between predicted and actual values.Also the MAE and RMSE values are greater than 0.5 indictaing poor predictions.However for Firefox 1.5 and 2.0 the KS values are between 0.2 and 0.5 which indicate a fair agreement between predicted and actual values.MAE values are near 0.3 which indicate slightly better results.Table2 shows that KS values are between 0.2 and 0.4 indicating a fair agreemnet between predicted and actual values.MAE is below 0.1 for all classifiers except SVM in which case it is 0.25.MAE values near to 0 indicate that most of the classifiers have accurately classified the test files.RMSE values are also below 0.3 which further validate the accuracy of results.Lower values of RMSE for RBF, D Table and ANN indicate that these classifiers are slightly more efficient than other classifiers.

Using the *isBuggy* criteria we correctly classified 75% of the header files except Firefox 1.5 for which the correct classification was below 20%.Using the *bugLevel* criteria, we correctly classified more than 95% of the header files. Table3 shows that for Firefox 1.0 and 1.5 KS values are negative which indicate that there is no agreement between predicted and actual values.MAE and RMSE values are greater than 0.7 for Firefox 1.5 indictaing very poor predictions.However for Firefox 2.0 the KS values are between 0.1 and 0.4 which indicate a fair agreement between predicted and actual values.MAE values are near 0.3 for Firefox 1.0 and 2.0 indicating slightly better results.Table4 shows that KS values are between 0.1 and 0.3 indicating a fair agreemnet between predicted and actual values.MAE is below 0.1 for all classifiers except SVM in which case it is 0.23.MAE values near to 0 indicate that most of the classifiers have accurately classified the test files.RMSE values are also between 0.1 and 0.3 which further validate the accuracy of results.Again lower values of RMSE for RBF, D Table,D Tree and ANN indicate that these classifiers are slightly more efficient than other classifiers.

Using the *isBuggy* criteria, we correctly classified more than 90% of the c files except Firefox 1.0 for which the correct classification was 72%. Table 5 shows that KS values are near to 0 indicating a poor agreemnet between predicted and actual values.MAE values are below 0.1 for Firefox 1.5 and 2.0 along with RMSE values below 0.3 indicating good results.However for firefox 1.0 results are poor as indicated by MAE and RMSE values.

For buglevel classification of cpp files D Table, RBF and SVM showed high values of CCI approaching 96%.NB was at the lowest level with average CCI value of 89%.D Tree, KNN and ANN showed intermediate CCI values as depicted in Figure2.

For buglevel classification of header files D Tree, D Table, RBF, ANN and SVM showed high values of CCI approaching 98%.NB was at the lowest level with average CCI value of 91%.KNN showed intermediate CCI values as depicted in Figure3.

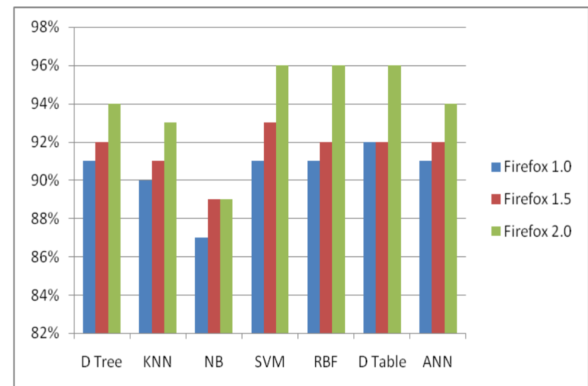

Figure 2. Classification of cpp files using buglevel criteria

5 Conclusions

All of the machine learning classifiers have shown almost similar results. Using the *bugLevel* criteria, we are able to classify the files with more than 90% accuracy. However, when using the *isBuggy* criteria, the results vary for different versions and can drop down to less than 50 %, which is unacceptable. One reason of this observation is that

Table 1. Classification of cpp files using isbuggy criteria

Classifier	Firefox 1.0				Firefox 1.5				Firefox 2.0			
	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE
DTree	39%	−0.36	0.62	0.74	77%	0.40	0.26	0.43	75%	0.27	0.28	0.46
KNN	37%	−0.40	0.63	0.80	77%	0.42	0.23	0.48	74%	0.28	0.26	0.51
NB	44%	−0.29	0.56	0.74	80%	0.40	0.20	0.44	81%	0.33	0.19	0.43
SVM	44%	−0.30	0.56	0.75	81%	0.44	0.19	0.43	82%	0.35	0.18	0.43
RBF	42%	−0.33	0.59	0.66	80%	0.43	0.29	0.39	79%	0.33	0.30	0.40
D Table	39%	−0.38	0.63	0.72	77%	0.40	0.28	0.41	76%	0.29	0.31	0.44
ANN	37%	−0.41	0.62	0.72	77%	0.42	0.26	0.39	75%	0.30	0.28	0.41

Table 2. Classification of cpp files using buglevel criteria

Classifier	Firefox 1.0				Firefox 1.5				Firefox 2.0			
	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE
D Tree	91%	0.28	0.07	0.23	92%	0.33	0.07	0.22	94%	0.28	0.05	0.18
KNN	90%	0.27	0.07	0.26	91%	0.35	0.06	0.24	93%	0.31	0.05	0.21
NB	87%	0.28	0.09	0.29	89%	0.36	0.08	0.27	89%	0.25	0.08	0.27
SVM	91%	0.08	0.26	0.33	93%	0.17	0.24	0.31	96%	0.25	0.23	0.29
RBF	91%	0.09	0.08	0.22	92%	0.14	0.07	0.20	96%	0.19	0.06	0.16
D Table	92%	0.21	0.07	0.22	92%	0.25	0.07	0.21	96%	0.32	0.05	0.16
ANN	91%	0.29	0.07	0.22	92%	0.33	0.07	0.21	94%	0.33	0.06	0.17

Table 3. Classification of header files using isbuggy criteria

Classifier	Firefox 1.0				Firefox 1.5				Firefox 2.0			
	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE
D Tree	65%	−0.11	0.39	0.55	16%	−0.09	0.76	0.82	85%	0.18	0.23	0.36
KNN	63%	−0.08	0.37	0.61	20%	−0.09	0.80	0.89	81%	0.13	0.18	0.43
NB	65%	−0.09	0.35	0.58	16%	−0.08	0.84	0.91	87%	0.23	0.13	0.35
SVM	72%	−0.04	0.28	0.53	12%	−0.03	0.88	0.94	95%	0.33	0.05	0.23
RBF	71%	−0.04	0.38	0.48	12%	−0.03	0.73	0.76	94%	0.28	0.23	0.28
D Table	68%	−0.09	0.40	0.51	12%	−0.07	0.75	0.78	91%	0.28	0.23	0.3
ANN	68%	−0.09	0.39	0.51	12%	−0.06	0.77	0.80	91%	0.29	0.21	0.28

Table 4. Classification of header files using buglevel criteria

Classifier	Firefox 1.0				Firefox 1.5				Firefox 2.0			
	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE
D Tree	96%	0.12	0.03	0.15	97%	0.10	0.03	0.13	98%	0.10	0.03	0.11
KNN	95%	0.22	0.03	0.17	96%	0.26	0.02	0.15	97%	0.18	0.02	0.15
NB	91%	0.21	0.06	0.24	91%	0.19	0.06	0.23	92%	0.16	0.06	0.23
SVM	96%	0.0	0.23	0.29	97%	0.0	0.23	0.28	98%	0.0	0.23	0.28
RBF	96%	0.12	0.04	0.15	97%	0.18	0.03	0.13	98%	0.14	0.03	0.11
D Table	96%	0.13	0.04	0.15	97%	0.12	0.03	0.13	98%	0.14	0.02	0.11
ANN	96%	0.11	0.03	0.15	97%	0.12	0.02	0.13	98%	0.07	0.02	0.11

Table 5. Classification of C files using isbuggy criteria

Classifier	Firefox 1.0				Firefox 1.5				Firefox 2.0			
	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE	CCI	KS	MAE	RMSE
D Tree	72%	0.0	0.28	0.52	91%	0.0	0.10	0.29	98%	0.0	0.03	0.13
KNN	73%	0.06	0.27	0.52	90%	0.04	0.10	0.31	97%	0.18	0.03	0.17
NB	67%	−0.06	0.33	0.56	85%	−0.02	0.16	0.38	92%	0.08	0.08	0.28
SVM	72%	0.0	0.28	0.53	91%	0.0	0.08	0.29	98%	0.0	0.02	0.13
RBF	72%	0.0	0.28	0.52	91%	0.0	0.10	0.29	98%	0.0	0.03	0.13
D Table	72%	0.0	0.28	0.52	91%	0.0	0.10	0.29	98%	0.0	0.03	0.13
ANN	72%	0.01	0.28	0.52	91%	0.01	0.09	0.29	98%	−0.01	0.03	0.14

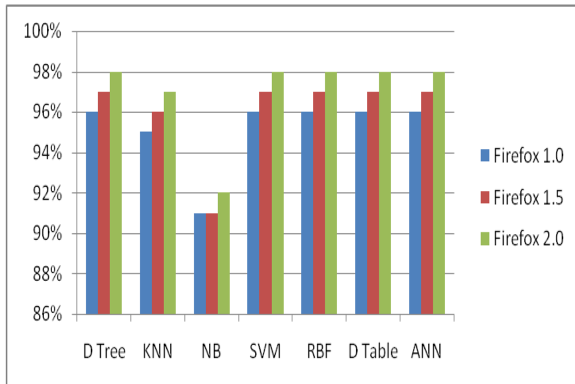

Figure 3. Classification of header files using bug level criteria

enhancements and bug corrections usually require a larger number of code changes.

In the future, we validate the applicability of machine learning techniques for prediction of bugs on other projects. We also want to answer the question whether it is possible to obtain a metrics or set of metrics that can be universally used as predictors of bugs in a variety of projects.

REFERENCES

[1] www.bugzilla.mozilla.org/
[2] www.cs.waikato.ac.nz/ml/weka/
[3] Aljahdali, S.H., Sheta, A., and Rine, D.,“Prediction of software reliability: a comparison between regression and neural network non-parametric models”, Computer Systems and Applications, ACS/IEEE International Conference on. 2001, 25-29 June 2001, Page(s): 470 -473.
[4] Boetticher, G.D. (2005). Nearest neighbor sampling for better defect prediction. ACM SIGSOFT Software Engineering Notes, 30(4), pp. 1-6, New York, NY: ACM Press.
[5] Y. Brun and M.D. Ernst,“Finding Latent Code Errors via Machine Learning over Program Executions”, Proc. 26th Int?l Conf. Software Eng., pp. 480-490, 2004.
[6] S. R. Chidamber and C. F. Kemerer, “A Metrics Suite for Object Oriented Design ”, IEEE Transactions on Software Engineering, 20(6), pp. 476-493, 1994.
[7] N.Fenton,and M. Neil, “A Critique of Software Defect Prediction Models ”, IEEE Transactions on Software Engineering, Vol. 25, No. 5, Sept. 1999.
[8] M. Fischer, Pinzger, M., Gall, H., “Populating a Release History Database from version control and bug tracking systems ”, Proceedings of International Conference on Software Maintenance, pp. 23-32, 2003.
[9] T. Gyimothy, R. Ferenc, and I. Siket, “Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction ”, IEEE Trans. Software Eng., vol. 31, no. 10, pp. 897- 910, Oct. 2005.
[10] S. Kim, E. J. Whitehead Jr., and Y. Zhang, “Classifying Software Changes: Clean or Buggy? ”, IEEE Transactions on Software Engineering, VOL. 34, NO. 2, pp. 181-196, MARCH/APRIL 2008.
[11] Koru, A.G. and Liu, H.,“Building effective defect-prediction models in practice”, Software, IEEE Volume 22, Issue 6, Nov.-Dec. 2005 Page(s):23 - 29
[12] N. Nagappan, T. Ball and A. Zeller,“Mining Metrics to Predict Component Failures”,Proceedings of the 28th international conference on Software engineering, November 2005,Shanghai,China.
[13] D.E.Neumann,“An Enhanced Neural Network Technique for Software Risk Analysis”, IEEE Transactions on Software Engineering, September, 2002.
[14] T.J. Ostrand, E.J. Weyuker, and R.M. Bell,“Predicting the Location and Number of Faults in Large Software Systems”, IEEE Trans. Software Eng., vol. 31, no. 4, pp. 340-355, Apr. 2005.
[15] A. Porter and R. Selby,“Empirically-guided software development using metric-based classification trees, IEEE Software”, Vol. 7, March 1990, pp. 46-54.
[16] M.Shepperd, and G. Kadoda,“Comparing software prediction techniques using simulation”, Software Engineering, IEEE Transactions on, Volume: 27 Issue: 11, Nov. 2001, Page(s): 1014 -1022.
[17] Venkata, U.B., B. Challagulla, B. Bastani Farokh, Y. I-Ling, 2005. Empirical Assessment of machine Learning based Software Defect Prediction Techniques. Proceedings of the 10th IEEE International Workshop on Object Oriented Real- Time Dependable Systems (WORDS?05), IEEE 2005.
[18] Ian H. Witten, Eibe Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann , June 2005
[19] Zhang, D., “Applying Machine Learning Algorithms in Software Development”, Modeling Software System Structures in a fastly moving Scenario, 2000 Monterey Workshop on, June 13 ? 16, 2000, Santa Margerita Ligure, Italy.
[20] T. Zimmermann, P. Weigerber, S. Diehl, and A. Zeller,“Mining Version Histories to Guide Software Changes”, IEEE Trans. Software Eng., vol. 31, no. 6, pp. 429-445, June 2005.