Validation of Software Quality Models using Machine Learning: An Empirical Study

Surbhi Gaur
Department of CSE/IT
GGSIPU, Dwarka

Savleen Kaur
Department of CSE/IT
GGSIPU, Dwarka

Inderpreet Kaur
Department of CSE/IT
GGSIPU, Dwarka

ABSTRACT
Software Quality is that significant nonfunctional requirement which is not fulfilled by many software products. In order to identify the faulty classes we can use prediction models using object oriented metrics. This paper empirically analyses the relationship between object oriented metrics and fault proneness of NASA Data sets using six machine Learning classifiers. It has been exhibited that Random Forest provides optimum values for accuracy, precision, sensitivity and specificity by performing Multivariate analysis of NASA Data sets.

General Terms
Software Maintenance, Software quality, confusion matrix, Accuracy, Precision, Recall, F-Measure, Sensitivity, Specificity, Bug

Keywords
Object-Oriented Software Metrics, Quality Metrics, Classifiers, ROC, Fault Proneness.

1. INTRODUCTION
Software maintenance is an activity that consumes lot of time and resource. In order to improve the effective cost as well as to achieve the customers' satisfaction and reliability of the software developed it is important to track the defect as early as possible in software life cycle.

Due to the intensification of complexity and constraints under which software is developed, it is becoming difficult to produce the software without faults. Because of the software failures the faulty classes may increase the development and maintenance costs and thus lead to reduction of reliability of the software [23].

Studies based on defect proneness have been conducted earlier [5,8-11, 14,15-20,22,27,29,33]. Software metrics are used in these studies [1, 4, 12, 13, 18, 24-26] for developing mathematical models to forecast the fault proneness. For verifying the capability of machine learning algorithms empirical validation of machine learning methods are required. To test any given hypothesis, clues that are collected from these empirical studies are treated to be dominant support.

In this paper software quality estimation using 6 machine learning classifies is executed. NASA datasets have been used for our empirical study [35].

The paper is formed as follows: Section 2 provides the related work done. Section 3 gives the metrics that are being taken used for analysis, with the research methodology followed in this study. Section 4 outlines the results of research. Conclusions made from the study are summarized in Section 5.

2. RELATED WORK
To identify model for forecasting fault proneness of classes [10] Briand et al. extracted 49 metrics. System that was scrutinized was medium sized c++ software system developed by undergraduate/graduate students. The eight systems that were under study consisted of 180 classes.To find individual and aggregate affect of OO metrics and fault proneness univariate and multivariate analysis were done. Results of the study demonstrated that all metrics were found to be substantial predictors of fault proneness where as NOC was insignificant predictor. In other study by Briand et al., commercial system with 83 classes was utilized where NOC was significant predictor of fault proneness and DIT metric was associated to fault proneness in inverse manner [8].

Chidamber and Kemerer metrics were affirmed by Gyimothy et al. on open source software. Linear and logistic regression neural and decision tree were utilized by them for model prediction. Their results proved that all other metrics except NOC were found significant predictors in LR analysis [17].

Based on study of open source agile software, WMC, CBO, RFC and LCOM metrics were considered to be very important as discovered by Olague et al., where as DIT was found insignificant in two versions of systems [29].

El Emam et al. found that class size has significant effect on fault proneness compared to other metrics by analyzing the effect of class size metric on defect proneness by taking telecommunication application [15].

Ramanath Subramanyam and M.S. Krishnan [33] have provided experimental proof that a subset of the Chidamber and Kemerer suite which are object oriented design complexity metrics plays a vital role in determining software defects.

Mohammad Alshayeb and Wei Li [3] provided short-cycled agile process and the long cycled framework evolution process as the two iterative procedures for the purpose of experimental study of object oriented metrics. Their results demonstrated that the design efforts and source lines of code added, changed, and deleted were not successfully anticipated in the long-cycled framework process but the same features were successfully anticipated by object oriented metrics in short-cycled agile process.

3. RESEARCH BACKGROUND
In this section, the metrics selected for this paper, empirical data collection, machine learning methods used, research hypothesis and measures used for the model are presented.

- **Metrics and Fault Proneness**

Fault proneness is defined as possibility of fault being present in the module. Our research deals with the metrics which are independent variables; the only binary dependent variableis fault pronenessThe effect of metrics is to be explored, on the fault proneness of a class, using machine learning methods.

- **Research Hypothesis**

In this section research hypothesis for this study is presented.

Hypothesis Set A

We tested the hypothesis given below to test the machine learning methods used.

H_o: There is no difference in accuracy of six machine learning classifiers.

H_A: At least one of the classifiers is more accurate than other.

- **Data Sets**

The data used in this research is collected from the NASA IV & V metrics Data Program. The main objective of Metrics Data Program is to collect, validate, organize, store and deliver the software metrics data. The data repository which consists of software metrics and associated error data, at the function/method level, can be accessed by the MDP. The data collected and validated by the MDP is stored and organized in the Data repository. The data in the data repository is for the NASA software development projects. The data is not generated by IV & V analysis [35].

The users are given an opportunity to find out the relationship among metrics or combinations of metrics to the software, with the help of the relation. The main role of the repository is to provide non-specific project data to the software community. The data is then cleaned, to make the data better, and authorized for publication through the MDP website and provided to the general users. The repository makes use of unparalleled numeric identifiers to identify single error records and product entries. The data is free of cost.

These are the different types of metrics [36]

Table 1.Types of Metrics for fault prediction

Type	**Rationale**	**Used By**
Process metrics	Bugs are caused by changes.	Moser
Previous defects	Past defects predict future defects	Kim
Entropy of changes	Complex changes are more errorprone than simpler ones.	Hassan
Churn (source code metrics)	Source code metrics are a better approximation of code churn	Novel
Entropy (source code metrics)	Source code metrics better describe the entropy of changes.	Novel

We have used Static code here because: Fenton collected information on how the software is to be built, what was built, and who build it, by dividing the metrics into process, product, and personnel. Static code measures do not display process and personnel details, but it is only a product metrics. Static code measures can be collected in a consistent manner across the projects. In the ideal manner, the companies which are CMM level 5, those companies carry out data mining, as the processes and the data collection is accurately stated. Ideally speaking, there is a existence of largely collected datasets, over many projects and many years. The datasets collected are in coherent form and there is no confusion in the nomenclature of the data.

The one element that can be approached in a connected way across many different projects is the source code. Even for very large systems, the feature of static code can be obtained, automatically and cheaply taken out. In Contrast, the methods such as manual code reviews require extensive skill. It depends on the review methods, as to 8 to 20 LOC/minute can be inspected. The same effort is put by other members of the review team, which can be as large as 4 or 6. [27].

Hence for all the above reasons, the static attributes are used, to guide software quality predictions, by researchers and industrial practitioners.

Datasets used in fault prediction systems may include metrics that are present in Table 2. The description is provided with it, in the next cell.

Table 2: Metrics Description

Metrics	**Description**	**Source**
WMC	Weighted methods per class (NOM: Number of Methods in the QMOOD metric suite)	C&K
5 N	total Operators + operands	Halstead
V	volume	Halstead
L	Length	Halstead
D	Difficulty	Halstead
I	Intelligence	Halstead
E	Effort	Halstead
B	Error	Halstead
DIT	Depth of Inheritance Tree	C&K
NOC	Number of Children	C&K
RFC	Response for a Class	C&K
CBO	Coupling between object classes	C&K
LCOM	Lack of cohesion in methods	C&K
LCOM3	LCOM Lack of cohesion in methods is a normalized version of the Chidamber and Kemerer's LCOM metric and its value varies between 0 and 2	Henderson-Sellers

IC	Inheritance coupling	quality oriented extension to C&ampK metric suite
CBM	Coupling Between Methods	quality oriented extension to C & amp K metric suite
AMC	Average Method Complexity	quality oriented extension to C & amp K metric suite
NPM	Number of Public Methods for a class(Also called as CIS: Class Interface Size)	QMOOD
DAM	Data Access Metric	QMOOD
MOA	Measure of Aggregation	QMOOD
MFA	Measure of Functional Abstraction	QMOOD
CAM	Cohesion among Methods of Class	QMOOD
CC	Cyclomatic complexity	McCabe's
LOC	Lines of Code	McCabe's
EV(G)	Essential complexity	McCabe's
IV(G)	design complexity	McCabe's
Ca	Afferent coupling	Martin's metrics
Ce	Efferent coupling	Martin's metrics

4. MACHINE LEARNING AND MODEL PREDICTION

Machine learning is a type of artificial intelligence that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data.Researchers are using machine learning, as better results are provided than regression and can incorporate complex nature of data .

Classification is technique in data mining which is used to predict group membership for data instances. Various classifiers used in this paper are summarized in Table 3.

Table 3: Classifiers and their Description

Classifier	**Description**
Naïve Bayes	A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model"[11].
Logistic Regression	Logistic regression is a type of regression analysis used for predicting the outcome of a categorical (a variable that can take on a limited number of categories) criterion variable based on one or more predictor variables. Logistic regression can be bi- or multinomial [20].
Instance Based(IB1)	It is Nearest-neighbor classifier and uses normalized Euclidean distance to find the training instance closest to the given test instance, and predicts the same class as this training instance. If multiple instances have the same (smallest) distance to the test instance, the first one found is used [2].
Bagging	Bagging is a ``bootstrap'' ensemble method that creates individuals for its ensemble by training each classifier on a random redistribution of the training set. Each classifier's training set is generated by randomly drawing, with replacement, N examples - where N is the size of the original training set; many of the original examples may be repeated in the resulting training set while others may be left out. Each individual classifier in the ensemble is generated with a different random sampling of the training set [7].
J48 Decision Tree	A decision tree is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. The internal nodes of a decision tree denote the different attributes; the branches between the nodes tell us the possible values that these attributes can have in the observed samples, while the terminal nodes tell us the final value (classification) of the dependent variable [28].
Random Forest	It is a learning ensemble consisting of a bagging of un-pruned decision tree learners with a randomized selection of features at each split [6].

5. MEASURES FOR MODEL VALIDATION

Specificity and Sensitivity validate the model's correctness. While specificity means proportion of classes predicted to be fault prone, sensitivity states the classes correctly predicted to be fault prone.

• Precision is the proportion of classes predicted correctly and F- measure is the harmonic mean on recall (sensitivity) and precision.

• Receiver operating characteristic (ROC) is plot of sensitivity on Y-coordinate and its specificity on x-coordinate. ROC is used effectively to evaluate the performance of prediction models [32]. Area under ROC Curve (AUC) is combined measure of sensitivity and specificity. To compute the accuracy of model being predicted, AUC is used [32].

6. EXPERIMENTS AND RESULTS

In this section we present the results of multivariate analysis of NASA IV & V metrics Data Program.The multivariate analysis tests the influence of multiple metrics on the fault proneness.

- **Multivariate Analysis**

Table 5 shows the multivariate analysis of the Test Project, under study where Random Forest with optimum AUC value provides a good model for study. Table 4 shows the model validation for Random Forest.

Table 4.Multivariate Analysis

Training Project	Modeling Technique	Accuracy	Precision	Recall	F-Measure	AUC	Specificity
CM1	**Naive Bayes**	0.82	0.89	0.91	0.90	0.69	0.33
	Logistic	0.85	0.94	0.90	0.92	0.73	0.21
	IB1	0.79	0.88	0.88	0.88	0.52	0.17
	Bagging	0.88	0.99	0.88	0.93	0.78	0.02
	J48	0.85	0.94	0.90	0.92	0.52	0.26
	Random Forest	0.84	0.94	0.89	0.91	0.72	0.12
JM1	**Naive Bayes**	0.81	0.95	0.84	0.89	0.68	0.21
	Logistic	0.82	0.98	0.83	0.90	0.71	0.10
	IB1	0.72	0.80	0.78	0.79	0.68	0.57
	Bagging	0.82	0.97	0.84	0.90	0.74	0.16
	J48	0.80	0.92	0.84	0.88	0.65	0.25
	Random Forest	0.80	0.91	0.86	0.88	0.73	0.32
MW1	**Naive Bayes**	0.82	0.85	0.94	0.89	0.73	0.56
	Logistic	0.89	0.96	0.92	0.94	0.65	0.26
	IB1	0.84	0.91	0.92	0.91	0.60	0.30
	Bagging	0.88	0.98	0.90	0.94	0.65	0.00
	J48	0.89	0.97	0.92	0.94	0.49	0.22
	Random Forest	0.91	0.97	0.92	0.95	0.75	0.30
PC1	**Naive Bayes**	0.88	0.93	0.94	0.94	0.77	0.36
	Logistic	0.92	0.98	0.93	0.96	0.83	0.18
	IB1	0.89	0.95	0.94	0.94	0.63	0.31
	Bagging	0.92	0.99	0.93	0.96	0.83	0.11
	J48	0.91	0.97	0.94	0.95	0.72	0.25
	Random Forest	0.91	0.97	0.93	0.95	0.79	0.13
PC2	**Naive Bayes**	0.95	0.96	0.99	0.98	0.88	0.31
	Logistic	0.98	0.99	0.99	0.99	0.78	0.06
	IB1	0.98	0.99	0.99	0.99	0.53	0.06
	Bagging	0.99	1.00	0.99	0.99	0.75	0.00
	J48	0.99	1.00	0.99	0.99	0.45	0.00
	Random Forest	0.99	1.00	0.99	0.99	0.69	0.00

PC3	**Naive Bayes**	0.36	0.28	0.95	0.44	0.74	0.89
	Logistic	0.87	0.97	0.89	0.93	0.82	0.20
	IB1	0.86	0.93	0.91	0.92	0.65	0.38
	Bagging	0.88	0.99	0.89	0.93	0.83	0.11
	J48	0.86	0.94	0.90	0.92	0.66	0.31
	Random Forest	0.86	0.94	0.91	0.92	0.79	0.31
MC1	**Naive Bayes**	0.94	0.94	1.00	0.97	0.91	0.62
	Logistic	0.99	1.00	0.99	1.00	0.88	0.28
	IB1	0.99	1.00	1.00	1.00	0.69	0.38
	Bagging	0.99	1.00	0.99	1.00	0.95	0.25
	J48	0.99	1.00	0.99	1.00	0.83	0.28
	Random Forest	1.00	1.00	1.00	1.00	0.87	0.47
MC2	**Naive Bayes**	0.73	0.92	0.74	0.82	0.72	0.39
	Logistic	0.76	0.83	0.80	0.82	0.71	0.61
	IB1	0.72	0.80	0.78	0.79	0.68	0.57
	Bagging	0.70	0.90	0.71	0.80	0.65	0.32
	J48	0.63	0.76	0.70	0.73	0.59	0.39
	Random Forest	0.61	0.76	0.68	0.72	0.64	0.32
PC4	**Naive Bayes**	0.87	0.94	0.91	0.93	0.83	0.38
	Logistic	0.91	0.97	0.93	0.95	0.91	0.48
	IB1	0.86	0.92	0.92	0.92	0.68	0.45
	Bagging	0.90	0.96	0.93	0.95	0.92	0.47
	J48	0.90	0.94	0.94	0.94	0.77	0.60
	Random Forest	0.89	0.95	0.93	0.94	0.90	0.48
PC5	**Naive Bayes**	0.96	0.98	0.98	0.98	0.94	0.46
	Logistic	0.97	0.99	0.98	0.99	0.96	0.29
	IB1	0.97	0.99	0.99	0.99	0.76	0.54
	Bagging	0.98	0.99	0.98	0.99	0.97	0.38
	J48	0.97	0.99	0.98	0.99	0.81	0.44
	Random Forest	0.98	0.99	0.99	0.99	0.80	0.51
KC1	**Naive Bayes**	0.82	0.91	0.89	0.90	0.791	0.37
	Logistic	0.86	0.98	0.87	0.92	0.80	0.22
	IB1	0.84	0.92	0.89	0.90	0.66	0.41
	Bagging	0.86	0.97	0.88	0.92	0.82	0.26
	J48	0.84	0.95	0.87	0.91	0.67	0.26
	Random Forest	0.85	0.93	0.89	0.91	0.80	0.38
KC3	**Naive Bayes**	0.79	0.88	0.86	0.87	0.64	0.33
	Logistic	0.80	0.90	0.86	0.88	0.71	0.33
	IB1	0.78	0.89	0.84	0.87	0.57	0.25
	Bagging	0.82	0.98	0.83	0.90	0.74	0.08
	J48	0.81	0.90	0.87	0.88	0.62	0.36
	Random Forest	0.79	0.90	0.85	0.87	0.72	0.28

Random forest gives the overall better results compared to other classifiers. AUC for random forest is 0.87. It gives

specificity of 0.47 while sensitivity is 1.00. Accuracy and precision are 1.00. ROC plot for multivariate analysis using top six classifiers for the dataset MC1 is given in Figure 1.

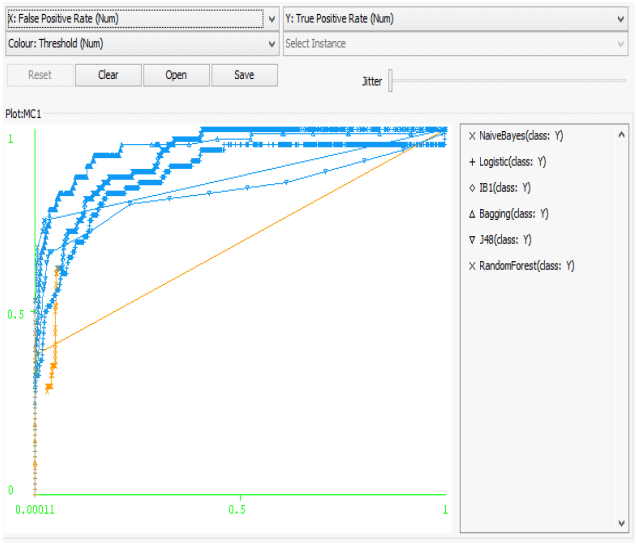

Figure 1: ROC plot for MC1

7. CONCLUDING REMARKS AND FUTURE WORK

The goal of our research is to empirically study performance of various classifiers on the data sets of NASA. Multivariate analysis performed on NASA datasets. We analyzed the metrics given by C&K, Martin, Handerson-seller, Mc-cabe and QMOOD.

Random Forest provided best ROC for most of the datasets. This study confirms that constructing Random Forest models for fault prediction is feasible and can be adapted for OO systems providing usefulness in predicting fault proneness.

More studies similar to this research may be conducted on different datasets to provide the generalized results for different organizations. Open Source projects are being now considered for fault prediction learning.We plan to replicate this study on other machine learning algorithms and focus on cost/benefit analysis to determine whether model would be economically possible.

8. REFERENCES

[1] Aggarwal, K.K., Singh, Y, Kaur, A. and Malhotra, R. 2006. Empirical Study of Object-Oriented Metrics, Journal of Object Technology, 5, 8.

[2] AHA, D. W. and Daniel, J. J. 1991. Instance-Based Learning Algorithms, Kluwer Academic Publishers, Boston. Manufactured in The Netherlands, Machine Learning, 6,37-66.

[3]Alshayeb M. and Li, W. 2003. An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes.IEEE transaction on software engineering, 12, 11, 1043-1049.

[4]Basili, V., Briand L. and Melo, W.L. 1996. A Validation of Object-Oriented Design Metrics as Quality Indicators.IEEE Transactions on Software Engineering, 22, 10,267-271.

[5]Bellini, P., Bruno, I., NesiandP. andRogai, D. 2005. Comparing fault-proneness estimation models", in Proc. of 10th IEEE International Conference on Engineering of Complex Computer Systems, 205–214.

[6]Breiman L. 2001. Random Forests. Machine Learning, 45, 1, 5-32.

[7]Breiman, L. 1996. Bagging Predictors. Machine Learning, 26, 123-140

[8]Briand, L. and Wust, J. 2001. Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs, Empirical Software Engineering: An International Journal , 6(1),11-58.

[9]Briand, L., Daly, J. and Wust, J. 1999. A Unified Framework for Coupling Measurement in Object-Oriented Systems, IEEE Transactions on software Engineering, 25, 91-121.

[10]Briand, L., Daly, J., Porter, V. and Wust, J. 2000. Exploring the relationships between design measures and software quality, Journal of Systems and Software, 5, 245-273.

[11]Briand, L., Melo, W.L. and Wust J. 2002. Assessing the applicability of fault-proneness models across object oriented software projects, IEEE Trans. on Software Engineering, vol. 28-7, 706–720.

[12]Chidamber, S. and Kemerer, C.F. 1994. A metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering.,SE-20, 6, 476-493.

[13]Chidamber, S. R. and Kemerer, C. F. 1991. Towards a metrics suite for object oriented design", in Proceedings of 6th ACM Conference on Object-Oriented Programming Systems Languagesand Applications (OOPSLA), Phoenix, Arizona, 197–211.

[14]Chidamber, S., Darcy, D. and Kemerer, C. 1998. Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis. IEEE Transactions on Software Engineering, 24, 8, 629-639. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[15]Emam, K. El, Benlarbi, S., Goel, N. and Rai, S. 2001. The Confounding Effect of Class Size on The Validity of Object-Oriented Metrics.IEEE Transactions on Software Engineering, 27, 7, 630-650.

[16]Emam, K. El, Melo, W. and Machado, J. 2001. The Prediction of Faulty Classes Using Object-Oriented Design Metrics, Journal of Systems and Software, 56, 63-75.

[17]Gyimothy, T., Ferenc, R. and Siket, I. 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Software Engineering, 31, 10, 897– 910.

[18]Henderson-sellers, B. 1996. Object-Oriented Metrics, Measures of Complexity, Prentice Hall, 1996 ISBN: 0-13-239872-9.

[19]Hitz, M. and Montazeri,B. 1995. Measuring Coupling and Cohesion in Object-Oriented Systems.Proc. Int. Symposium on Applied Corporate Computing, Monterrey, Mexico.

[20]Hosmer, D. W.andLemeshow, S. Applied Logistic Regression ISBN: 9780471356325.

[21]Huang, K. 2003. Discriminative Naive Bayesian Classifiers, Department of Computer Science and Engineering, the Chinese University of Hong Kong.

[22]Khoshgoftaar, T.M. and Seliya, N. 2004. Comparative assessment of software quality classification techniques: An empirical study. Empirical Software Engineering, 9, 229–257.

[23]Koru, A. and Liu, H. 2005. Building effective defect prediction models in practice, IEEE Software,.23–29

[24]Li, W. and Henry, S. 1993. Object Oriented Metrics that Predict Maintainability. Journal of Systemsand Software, 23, 2,111-122.

[25]Lorenz, M. and Kidd,J. 1994. Object-Oriented Software Metrics.Prentice-Hall.

[26]McCabe & Associates, McCabe Object Oriented Tool User's Instructions, 1994.

[27]Menzies, T., Greenwald, J. and Frank, A. 2007. Data mining static code attributes to learn defect predictors. IEEE Trans.on Software Engineering, 33, 1, 2–13.

[28]Mitchell, T. 1997 Machine Learning, ISBN 0070428077, McGraw Hill, 1997,available at http://www.cs.cmu.edu/~tom/mlbook-chapter-slides.html

[29]Olague, H., Etzkorn ,L., Gholston, S. and Quat-tlebaum S. 2007 Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. IEEE Transactions on software Engineering, 33, 8, 402-419.

[30]Rosenberg, L. and Hyatt, L. 1995. Software Quality Metrics for Object Oriented System Environments, NASA Technical Report.

[31] Schroeder, M. 1999. A practical Guide to Object-Oriented Metrics, IT Professional,1-6, 30-36.

[32]Stone, M. 1974. Cross-validatory choice and assessment of statistical predictions, J. Royal Stat. Soc., 36,111-147.

[33]Subramanyam, R. and Krishnan, M.S. 2003. Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. IEEE transaction on software engineering, 29, 4, 297-310.

[34]Witten, IH. and Frank, E. 2005. Data mining: practical machine learning tools and techniques. Morgan Kaufmann

[35] http://nasa-softwaredefectdatasets.wikispaces.com/

[36] M.D'Ambros, M.Lanza and R. Robbes, "Evaluating Defect Prediction Approaches: A Benchmark nd an Extensive Comparison," Empir Software Eng , Vol. 17,2012, pp. 531 -577, DOI 10.1007/s10664-011-9173-9.