International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161
©2014 INPRESSCO®, All Rights Reserved
Available at http://inpressco.com/category/ijcet

General Article

Fault Prediction Modeling using Object-Oriented Metrics: An Empirical Study

Navpreet Kaur[Á*] and Aman Paul[Á]

[Á]Department of Computer Science, Eternal University, Baru Sahib, Sirmour (HP)

Accepted 30 January 2014, Available online 01 February 2014, **Vol.4, No.1 (February 2014)**

Abstract

Software testing is an area where software products are examined through a series of verification and validation processes respectively. This phase of software development carries out the process of detection and removal of software faults. But this detection and removal of faults together consume up to 60% of project budget ***(Beizer, 1990).*** *Applying equal testing and verification efforts to all parts of a software system becomes cost-prohibited. Software Fault Proneness is a key factor for monitoring and controlling the quality of software. By comparing the distribution of faults (Fault Proneness) and the amount of faults found with testing (Software faultiness), the effectiveness of analysis and testing can be judged easily. Detecting a fault prone code early, within software life-cycle phase, allows for the code to be fixed at minimum costs; thus a good fault prediction helps to lower down the development and maintenance costs. In this project, software quality estimation, using various classifiers is performed where some metrics are the inputs and its quality attributes bring the output. Empirical study of these classifiers then judges the quality of the software being developed.*

Keywords: *Software Testing, Classification Algorithms, bugs, Weka 3.6.9 tools, Performance analysis.*

1. Introduction

The demand for software is growing whereby the use of software is becoming more common in every field around the world. Software engineering (SE) is an area where a developer after collecting the user requirements, designs the software using systematic, disciplined and quantifiable approaches. In layman's terms, it is the act of conceiving, modeling and drawing a solution to a problem undertaken **(Laplante and Phillip, 2007).**While developing software, the software developer may make mistakes due to which there is going to be a certain amount of faults/errors in that software giving annoying outputs or may be a complete system crash. Thus to have an efficient quality software, the careless faults introduced by the developer need to be detected and hence removed from the software. According to Beizer, the detection and removal of faults together can consume up to 60% of a project budget **(Beizer, 1990).** By comparing the distribution of faults (Fault Proneness) and the amount of faults found with testing (Software faultiness), the effectiveness of analysis and testing can be easily judged. Detecting a fault prone code early, within software life-cycle phase, allows for the code to be fixed at lower costs; thus a good fault prediction helps to lower down the development and maintenance costs. Im my research an Open source java project is taken where SOURCEFORGE keeps the track of bugs encountered along with their status. A tool will then calculate the metrics for the open source projects and then WEKA is used for finding best modeling classifier among various classifiers used. The independent variables include other CK metrics like DIT, NPM, RFC, CBO, etc whereas the dependent variables represent the faultiness of the dataset. Different measures including accuracy, precision, recall or sensitivity, etc are evaluated and based on them; an empirical study for the project using different classifiers is performed.

*Corresponding author: **Navpreet Kaur**

2. Classification

Classification is a data mining (machine learning) technique used to predict group membership for data instances. For example, you may wish to use classification to predict whether the weather on a particular day will be “sunny”, “rainy” or “cloudy”.

Classifiers used in this research are as follows

- **Bayes Net:** Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available. A Bayesian network, Bayes network, belief network, Bayesian model or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). **(Ben-Gal I., et al.. 2007).**

- **Naive Bayes:** A naive Bayes classifier is a simple probabilistic classifier based on applying Baye's theorem with strong (naive) independence assumptions. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable **(K. Huang, et al.. 2003).**
- **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. Binomial or binary logistic regression refers to the instance in which the criterion can take on only two possible outcomes (e.g., "dead" vs. "alive", "success" vs. "failure", or "yes" vs. "no"). **(D. W. Hosmer, et al.. 2005).**
- **RBF Network:** A radial basis function network is an artificial neural network that uses radial basis functions as activation functions. It is a linear combination of radial basis functions. They are used in function approximation, time series prediction, and control. Radial basis function (RBF) networks typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer **(T. Mitchell, 1997).**
- **Multilayer perceptron:** Multi Layer perceptron (MLP) is a feed forward neural network with one or more layers between input and output layer. Feed forward means that data flows in one direction from input to output layer (forward). This type of network is trained with the back propagation learning algorithm. MLPs are widely used for pattern classification, recognition, prediction and approximation. Multi Layer perceptron can solve problems which are not linearly separable. A multilayer perceptron (MLP) is a feed forward artificial neural network model that maps sets of input data onto a set of appropriate output. **(N. Leonardo, Tutorial).**
- **SMO:** Sequential minimal optimization (SMO) is an algorithm for efficiently solving the optimization problem which arises during the training of support vector machines. It was invented by John Platt in 1998 at Microsoft Research.SMO is widely used for training support vector machines (N. Cristianini, 2000). SMO is an iterative algorithm for solving the optimization problem.SMO breaks this problem into a series of smallest possible sub-problems, which are then solved analytically
- **K-star:** K-Star is an instance-based classifier that is the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function **(DAVID W. AHA, 1991).**
- **Instance based (IB1):** It is Nearest-neighbour 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 **(L. Breiman, 1996).**
- **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 **(L. Breiman, et al..1996).**
- **Decision Table:** Decision Tables are simple supervised classifiers which, Kohavi demonstrated, can outperform state-of-the-art classifiers such as C4.5. Decision Table can create more queries; it is more of multipath/ multiflow (V. J. Hodge, Tom Jackson).
- **Random Forest:** A random forest is a classifier consisting of a collection of tree structured classifiers {h(x,Èk), k=1, ...} where the {Èk} are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x .Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges as to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them **(BreimanL,et al..2001).**
- **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 (V. J. Hodge).

3. Related Work

The earlier studies in defect prevention were focused on defect prediction and decide upon the team size of the testing resources required in order to complete the project on time and lot of efforts were utilized in the debugging and get the defects eliminated. With the advent of SDLC processes many companies formulated their own defect prevention mechanisms and many studies were conducted towards defect prediction and prevention. In the software industry, the defect density of a software system is usually measured only after the implementation of the project through code. Basili and Warmer emphasize that an early warning about the probable defect levels in the system will benefit the team in software development process and promote better defect management strategies **(Basili, et al., 1996).**

Based on a study of eight medium-sized systems, developed by students Basili found that several of the metrics were associated with fault proneness. Briand et al empirically explored the relationship between OO metrics and the probability of fault detection in system classes. Their results indicated that very accurate prediction models could be derived to predict faulty classes **(Laplante and Phillip, et al., 2007).**

As in 1999, 49 metrics were extracted to identify model for predicting fault proneness of classes. System that was investigated was medium sized C++ software system developed by undergraduate/graduate students. The eight systems under study had total 180 classes. Univariate and multivariate analysis was done to find individual and combined impact of OO metrics and fault proneness. Result of the study showed that all metrics except NOC to be significant predictor of fault proneness. In other study mentioned in **(B.Henderson-sellers, et al..1996),** commercial system with 83 classes was used where DIT metric was related to fault proneness in inverse manner and NOC was significant predictor of fault proneness **(L. Briand et al.,1999).**

Based on study of open source agile software, it was found that WMC, CBO, RFC and LCOM metrics to be very significant while DIT was found insignificant in two versions of systems **(M. Alshayeb and W. Li, et al..2003).**

In 2001, authors studied the effect of class size metric on defect proneness by taking telecommunication application and found that class size has significant effect on fault proneness compared to other metrics **(Emam et al..2001).**

Gyimothy validated Chidamber and Kemerer metrics on open source software .They employed linear and logistic regression neural and decision tree for model prediction. Their results showed that NOC was not a significant predictor whereas all other metrics were found significant in LR analysis **(Gyimothy ,et al.2005).**

El Emam studied the effect of class size metric on defect proneness by taking telecommunication application and found that class size has significant effect on fault proneness compared to other metrics **(Emam et al..2001).**

Datasets and tool used

1. Hardware: We conduct our evaluation on Intel Pentium P6200 platform which consist of 1 GB memory and 320 GB hard disk.

2. Tools used:

a) Source Forge
b) ckjm_ext_20
c) Weka-3-6-9

Source Forge

Source forge helps tracking bugs from its tracker part. Their status is known with its patches made respectively. Open source java project that has been tracked using source forge with its summary is given below:

➢ JFreeChart

JFreeChart is an open-source framework for the programming language Java, which allows the creation of a wide variety of both interactive and non-interactive charts. JFreeChart supports a number of various charts, including combined charts (http://www.jfree.org/jfreechart):

- X-Y charts (line, spline and scatter). Time axis is possible.
- Pie charts
- Gantt charts
- Bar charts (horizontal and vertical, stacked and independent). It also has built-in histogram plotting.
- Single valued (thermometer, compass, speedometer) that can then be placed over map.
- Various specific charts (wind chart, polar chart, bubbles of varying size, etc.).

It is possible to place various markers and annotations on the plot. JFreeChart also works with GNU Classpath, a free software implementation of the standard class library for the Java programming language. JFreeChart automatically draws the axis scales and legends. Charts in GUI automatically get the capability to zoom in with mouse and change some settings through local menu. The existing charts can be easily updated through the listeners that the library has on its data collections.

➢ CKJM TOOL

ckjm — Chidamber and Kemerer Java Metrics.

The program *ckjm* calculates Chidamber and Kemerer object-oriented metrics by processing the byte code of compiled Java files. The program *CKJM extended* calculates nineteen size and structure software metrics by processing the byte code of compiled Java files.

Metrics evaluated by CKJM

- **Weighted Methods per Class (WMC):**

This measures the sum of complexity of the methods in a class. A predictor of the time and effort required to develop and maintain a class we can use the number of methods and the complexity of each method. A large number of methods in a class may have a potentially larger impact on the children of a class since the methods in the parent will be inherited by the child.

- **DIT- Depth of Inheritance Tree**

DIT metric is used to find the length of the maximum path from the root node to the end node of the tree. DIT represents the complexity and the behavior of a class, and the complexity of design of a class and potential reuse. Thus it can be hard to understand a system with many inheritance layers. On the other hand, a large DIT value indicates that many methods might be reused.

- **NOC - Number of Children**

A class's number of children (NOC) metric simply measures the number of immediate descendants of the class.

- **CBO - Coupling between object classes**

The coupling between object classes (CBO) metric represents the number of classes coupled to a given class (efferent couplings, Ce). This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.

- **RFC - Response for a Class**

The response set of a class (RFC) is defined as set of methods that can be executed in response and messages received a message by the object of that class. Larger value also complicated the testing and debugging of the object through which, it requires the tester to have more knowledge of the functionality. The larger RFC value takes more complex is class is a worst case scenario value for RFC also helps the estimating the time needed for time needed for testing the class.

- **LCOM - Lack of cohesion in methods**

This metric is used to count the number of disjoints methods pairs minus the number of similar method pairs used. The disjoint methods have no common instance variables in the methods, while the similar methods have at least one common instance variable. It is used to measuring the pairs of methods within a class using the same instance variable.

- **Ca - Afferent couplings**

A class's afferent coupling is a measure of how many other classes use the specific class. Ca is calculated using the same definition as that used for calculating CBO (Ce).

- **NPM - Number of Public Methods**

The NPM metric simply counts all the methods in a class that are declared as public. It can be used to measure the size of an API provided by a package.

- **Ce - Efferent couplings**

A class's efferent coupling is a measure of how many other classes is used by the specific class. Coupling has the same definition in context of Ce as that used for calculating CBO.

- **LCOM3 - Lack of cohesion in methods.**

LCOM3 varies between 0 and 2.

- **LOC - Lines of Code.**

The lines are counted from java binary code and it is the sum of number of fields, number of methods and number of instructions in every method of given class.

- **DAM: Data Access Metric**

This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class. A high value for DAM is desired. (Range 0 to 1).

- **MOA: Measure of Aggregation**

These metric measures the extent of the part-whole relationship, realized by using attributes. The metric is a count of the number of data declarations (class fields) whose types are user defined classes.

- **MFA: Measure of Functional Abstraction**

This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class. The constructors and the java.lang.Object (as parent) are ignored. (Range 0 to 1).

- **CAM: Cohesion among Methods of Class**

This metric computes the relatedness among methods of a class based upon the parameter list of the methods. A metric value close to 1.0 is preferred. (Range 0 to 1).

- **Inheritance Coupling**

This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods functionally dependent on the new or redefined methods in the class.

- **CBM: Coupling Between Methods**

The metric measure the total number of new/redefined methods to which all the inherited methods are coupled. There is a coupling when one of the given in the IC metric definition conditions holds.

- **AMC: Average Method Complexity**

This metric measures the average method size for each class. Size of a method is equal to the number of java binary codes in the method.

- **CC - The McCabe's Cyclomatic complexity**

It is equal to number of different paths in a method (function) plus one. The cyclomatic complexity is defined as:**CC = E - N + P**. Where, E - Number of edges of the graph, N -Number of nodes of the graph, P - Number of connected components

Operation of CKJM

To work in windows with ckjm tool, the steps are as follows:
(http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/oper.html):

1. Run the following command the ckjm tool with to calculate metrics of only one class:
Path of ckjmjar:\> java -jar\path of ckjm\ ckjm_ext_20.jar path of classfile.class

2. The command's output will be a list of class names (prefixed by the package they are defined in), followed by the corresponding metrics for that class: WMC, DIT, NOC, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LOC, DAM, MOA, MFA, CAM, IC, CBM and AMC . In lines where at the beginning is "~" is the value of CC. CC is counted for all methods in given class.If you specify neither jars nor classes to analyze ckjm will prompt you to type some.

3. Calculating each class metrics is time consuming so save the class files in one folder and run the command. for %f in (*.class) do java -jar c:pathoffolder\pathofckjms\-x pathofclassfilefolder %f>>classfilefoldername.text

4. The ck metrics evaluated can be directly saved into a txt file and then imported into excel.

- **Weka tool:** In this experiment, we used Weka 3.6.9 and window 8 to evaluate the performance of classification algorithms using time taken to build the model according to respective no of clusters. Weka is machine learning/data mining software written in Java language (distributed under the GNU Public License).

Waikato Environment for Knowledge Analysis
Version 3.6.9
(c) 1999 - 2013
The University of Waikato
Hamilton, New Zealand

Fig 1: Weka Application

Weka is a collection of machine learning algorithms for data mining tasks. Weka contains tools for developing new machine learning schemes. It can be used for Pre-processing, Classification, Clustering, Association and Visualization.

4. Terms used

Confusion matrix

In the field of machine learning, a confusion matrix is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class.

Table 1: Confusion Matrix

		Predicted	
		Negative	Positive
Actual	Negative	**a**	**b**
	Positive	**c**	**d**

The entries in the confusion matrix have the following meaning in the context of our study:
a is the number of correct predictions that an instance is negative, b is the number of incorrect predictions that an instance is positive, c is the number of incorrect of predictions that an instance negative, d is the number of correct predictions that an instance is positive.

Accuracy: The accuracy (AC) is the proportion of the total number of predictions that were correct. It is determined using the equation: **Accuracy = (a+d)/(a+b+c+d)**
Recall or Sensitivity: The recall or true positive rate (TP) is the proportion of positive cases that were correctly identified, as calculated using the equation: **Recall = d/(c+d)**
False Positive rate: The false positive rate (FP) is the proportion of negatives cases that were incorrectly classified as positive, as calculated using the equation: **FPR =b/(a+b)**
True Negative Rate: The true negative rate (TN) is defined as the proportion of negatives cases that were classified correctly, as calculated using the equation: **TNR =a/(a+b)**
False Negative Rate: The false negative rate (FN) is the proportion of positives cases that were incorrectly classified
as negative, as calculated using the equation: **FNR = c/(c+d)**
Precision: The precision (P) is the proportion of the predicted positive cases that were correct, as calculated using the following equation: **P = d/(b+d)**
F –measure
A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional F-measure. **F-measure= 2 * ((Precision. Recall) / (Precision + Recall))**
ROC: In signal detection theory, a receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives **(TPR= true positive rate)** vs. the fraction of false positives out of the negatives **(FPR = false positive rate)**, at various threshold settings. **(TPR is also known as sensitivity and FPR is one minus the specificity or true negative rate.)**
The ROC is also known as a **Relative Operating Characteristic curve**, because it is a comparison of two operating characteristics **(TPR & FPR)** as the criterion changes.

5. Experiment Result and Discussion

The application of JFreeChart consisted of 1056 classes over which a cross validation with 10 fold was performed. The multivariate analysis predicted the combined effect of all metrics on fault proneness. Table 5.1.2.1 shows the multivariate analysis of JFreeChart. The results show that Random Forest with ROC (0.863), provides a better prediction measures and multivariate analysis out performs the univariate analysis in prediction of fault proneness.

Multivariate Analysis of JFreeChart

Training Project	Test Project	Modeling Technique	Accuracy	Recall (Sensitivity)	False(+)ve Rate	True(-)ve Rate	False(-)ve Rate	Precision	F-Measure	ROC Area
JFreeChat	Cross-validation	Bayes Net	0.94	0.32	0.029	0.97	0.68	0.35	0.33	0.843
		Naive Bayes	0.89	0.38	0.08	0.91	0.62	0.19	0.25	0.721
		Logistic	0.95	0.34	0.015	0.98	0.66	0.53	0.41	0.797
		SMO	0.95	0.00	0.00	1.00	1.00	0.00	0.00	0.5
		MLP	0.95	0.40	0.015	0.98	0.60	0.55	0.46	0.831
		RBF	0.95	0.00	0.00	1.00	1.00	0.00	0.00	0.701
		IB1	0.93	0.44	0.036	0.96	0.56	0.37	0.46	0.702
		KStar	0.95	0.4	0.021	0.97	0.60	0.47	0.43	0.845
		Bagging	0.96	0.26	0.003	0.99	0.74	0.81	0.39	0.79
		Random Forest	0.96	0.62	0.009	0.99	0.62	0.65	0.63	0.863
		J48	0.97	0.56	0.009	0.99	0.44	0.75	0.64	0.716

Table 2: Observations

Rank allotments for Multivariate Analysis

Classifier	Rank
Random Forest	1
J48	2
MLP	3

Random Forest with accuracy (0.96), Sensitivity (0.62) and precision (0.65) thus can lead to development of optimum software quality models for predicting the fault prone classes. Confusion matrix for Random Forest, evaluated by weka is given in table 2.

Table 3: Model Validation of JFreeChart

		Predicted	
		Not faulty	Faulty
Observed	Not Faulty	996	10
	Faulty	31	19

ROC for True and false classes of PMD by Random Forest are presented in fig. 2 and 3 respectively:

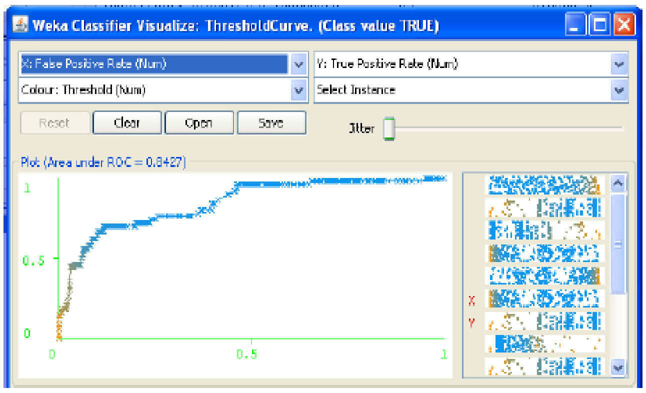
Weka Classifier Visualize: ThresholdCurve. (Class value TRUE)
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot (Area under ROC = 0.8427)
1
0.5
0
0
0.5
1
X
Y

Fig 2: ROC curve for True Classes of JFreeChart

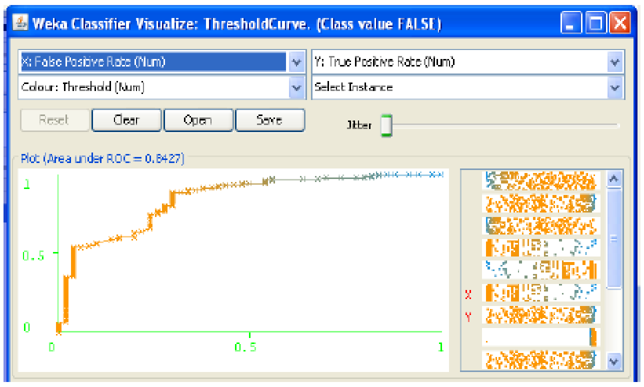
Weka Classifier Visualize: ThresholdCurve. (Class value FALSE)
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot (Area under ROC = 0.8427)
1
0.5
0
0
0.5
1
X
Y

Fig 3: ROC curve for False Classes of JFreeChart

Defect proneness results of JFreeChart

The goal of our research is to empirically study performance of various classifiers on the data set given by JFreeChart. We find the individual and combined effects of each metric were well predicted using 12 machine learning techniques. We analyzed the metrics given by C&K, Martin, Handerson-seller, Mc-cabe and QMOOD. Our main results are as follows:

- DIT has highest value of sensitivity and precision. AMC, Ca, CBM, DAM, IC, LCOM3, LOC, Max_CC, MOA, NOC and NPM metrics predicted all classes to be non-faulty. Therefore, the results of 12 machine leaning method show that AMC, Ca, CBM, DAM, IC, LCOM3, LOC, Max_CC, MOA, NOC and NPMare poor predictors of fault proneness.
- Random Forest provided good AUC using ROC analysis. 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.

Conclusions and Future Scope

Open Source software are freely available and any type of user may carry out testing operations on any available dataset to estimate the quality of software. Our focus is to determine whether model would be economically viable or not. More studies similar to this research may have been conducted on different datasets to provide the generalized results for different organizations. Focus here is to perform this study on other machine learning algorithms and focus on cost/benefit. 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 of the application undertaken.

References

Alan O, C atal C, (2010) Thresholds based outlier detection approach for mining class outliers: An empirical case study on software measurement datasets. The Scientific and Technological Research Council of urkey (TUBITAK), The National Research Institute of Electronics and Cryptology (UEKAE), Information Technologies Institute,41470 kocaeli, Turkey, 3440-3445 B.Henderson-sellers (1996) Object-Oriented Metrics, Measures of Complexity. Prentice Hall,ISBN:0-13-239872-9

BreimanL (2001) Random Forests. Machine Learning, 45:5-32

Ben-Gal I., Bayesian Networks, in Ruggeri F., Faltin F. & Kenett R., Encyclopedia of Statistics in Quality & Reliability, Wiley & Sons (2007) at http://www.eng.tau.ac.il/~bengal/BN.pdf ConfusionMatrix, http:// www2.cs. uregina.ca/~dbd/cs831/notes/ confusion_ matrix/confu sion_matrix.html

David W. AHA and Jody J. Daniel (1991) Instance-Based Learning Algorithms,Kluwer

Academic Publishers, Boston. Manufactured in The Netherlands,Machine Learning, 6:37-

D. W. Hosmer and Stanley Lemeshow (2005) Applied Logistic Regression, ISBN: 9780471356325

Elena P´erez-Mi˜nana and Jean-Jacques Gras (2006) Improving fault prediction using bayesian networks for the development of embedded software applications. Software Testing, Verification Reliability, 16:157–174

http://www.jfree.org/jfreechart

http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/intro.html

http://www.spinellis.gr/sw/ckjm/doc/metric.html

http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/oper.html

http://www.gtbit.org/downloads/dwdmsem6/dwdmsem6lman.pdf

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

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

K. Ganesan, Taghi M. Khoshgoftaar, and Edward B. Allen(2000) Verifying requirements through mathematical modelling and animation. In International Journal of Software Engineering and Knowledge Engineering, 10:139–152

K. Huang (2003) Discriminative Naive Bayesian Classifiers, Department of Computer Science and Engineering, The Chinese University of Hong Kong, 1-21

K.K.Aggarwal, Y. Singh, A.Kaur and R.Malhotra (2006) Empirical Study of Object-Oriented

Metrics.In Journal of Object Technology, 5:1-20

L. Breiman (1996) Bagging Predictors. Machine Learning , 26:123-140

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

L.Briand, J.Daly, V. Porter and J. Wust (2000) Exploring the relationships between design measures and software quality. In Journal of Systems and Software, 5: 245-273

L.C. Briand, W.L.Melo and J. Wust (2002) Assessing the applicability of fault-proneness models across object oriented software projects.IEEE Trans. on Software Engineering, 28:706–720

L. Guo, Y. Ma, B. Cukic, and H. Singh (2004) Robust prediction of fault-proneness by random forests. In Proceedings of the 15nd International Symposium on Software Reliability Engineering