Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 11; April-June, 2015 pp. 30-35
© Krishi Sanskriti Publications
http://www.krishisanskriti.org/acsit.html

Prediction of Change Prone Classes using Threshold Methodology

Ruchika Malhotra[1] and Ankita Bansal[2]

[1,2]Dept. of Software Engineering Delhi Technological University, Delhi, India
E-mail: [1]ruchikamalhotra2004@yahoo.com, [2]ankita.bansal06@gmail.com

Abstract—*The widespread use of software metrics to predict various quality attributes is evident. In this study we have used metrics to identify change prone parts of software so that developers can pay focused attention on such classes. Various metric models are constructed using machine learning and statistical techniques, which can be used for predicting change prone parts of the software. However, training these models is a time consuming task and hence, these models cannot be used on a daily basis to predict change proneness. In this paper, an alternative approach is used which is based on calculating thresholds of metrics. Thresholds are defined as alarming values above which a class is considered to be risky or change prone and hence, needs careful attention. A statistical approach is used to calculate threshold values of open source software, Freemind 0.9.0. To examine the applicability of threshold values, they are validated on the different releases of Freemind as well as on a similar nature project, Frinika. The results demonstrate the effectiveness of the methodology in identification of the threshold values.*

Keywords*: Software quality, threshold, metrics, logistic regression, empirical validation*

1. INTRODUCTION

Object oriented (OO) metrics are widely used in literature to predict various software quality attributes. In this study, we use OO metrics to predict the parts of software that are more change prone than others. Identification of change prone parts in early phases of software development life cycle helps managers is efficiently allocating the resources (time, money and manpower). It also helps designers to get an insight on the design of software and thus, can alter the design if it is required. Construction of the metric models using various machine learning and statistical techniques to identify change prone parts is well known in literature ([1]-[7]). However, construction of these metric models is not always feasible. Training these models using machine learning techniques is a time consuming task and thus, it is impractical to use them on a daily basis. An alternative is to identify certain alarming values of the metrics above which a class is considered to be risky. In other words, we can define some thresholds for the metrics. The class having metric value more than the threshold value needs careful and focused attention. According to Chidamber et al. [8] one of the important uses of OO metrics is to identify extreme values (threshold values) of these metrics. The classes above these extreme values will have higher complexity and require management attention. In this study, we have obtained threshold values for various OO metrics using a statistical approach proposed by Bender [9]. This methodology of threshold computation is much simpler as it is just based on the values of constant and coefficient of each metric obtained from univariate logistic regression. It requires much less time and effort than the traditional metric models.

In initial years, many authors have derived threshold values based on their experience and thus, those values are not universally accepted. For example, McCabe [10] defined a value of 10 as threshold for the complexity metric; the threshold values of maintainability index metric are defined as 65 and 85 [11] etc. Besides the thresholds based on intuition, some researchers defined thresholds using mean (μ) and standard deviation (σ) measures. For example, Erni et al. [12] calculated the threshold values using μ and σ of the metric values. The paper defined two terms Tmin = m - s and Tmax = m + s, the lower and the higher thresholds. However, this method did not become popular as it was based on the assumption that the metrics are normally distributed which is not always possible. French [13] used Chebyshev's inequality theorem (not restricted to normal distribution) in addition to mean (μ) and standard deviation (σ) to derive threshold values. According to French, a threshold can be defined as $T = \mu + k * \sigma$ (k=number of standard deviations). However, this methodology was also not used much as it was restricted to only two-tailed symmetric distributions, which are not justified. There are few studies in literature, which have obtained the threshold values of metrics to predict fault proneness ([14]-[20]). Benlarbi et al. [21] and El Emam et al. [19] used a statistical model (based on logistic regression) suggested by Ulm [22] to calculate the threshold values of number of metrics and found that there was no statistical difference between the model predicted with threshold values and model predicted without threshold values. Bender [9] working in the epidemiological field proposed another statistical model that was used by Shatnawi [20] to find the risk level for any arbitrary threshold value for Chidamber and

Kemerer (CK) metrics. Their results concluded that the CK metrics have threshold effects at various risk levels.

Thus, the literature shows that although there are few studies that use threshold methodology for predicting fault proneness, but no study till date known to the authors has obtained thresholds for OO metrics on change proneness. More studies need be conducted to identify potential usage of threshold values for predicting change proneness. Additionally, research is required to understand if these values can be generalised across various software applications.

Hence, the main aim of this paper is to calculate the threshold values of metrics and compare its results of validation with the results of the validation when metric models are used. We focus on the following four key points:

1. Applying threshold methodology to compute thresholds: We have used a statistical approach proposed by Bender et al. [9] to calculate the threshold values of the metrics of an open source software, Freemind (a mind mapper and hierarchical editor).
2. Validating the threshold methodology to assess its accuracy: Once the threshold values are computed, we constructed various machine learning models (adaboost (AB), bagging, logitboost (LB), multilayer perceptron (MLP), naïve bayes (NB), random forest (RF) and classification & regression trees (CRT)) to validate the values. We have also constructed the same models without using the thresholds of the metrics and compared the results.
3. Conducting inter-release validation: We computed the threshold values for Freemind 0.9.0 and validated them on different releases of Freemind, 0.9.1 and 0.10.0. This will allow us to assess the applicability of the threshold values on various versions of the same software.
4. Conducting inter-project validation: In addition to validating the different release of same software, we also validated the threshold values on similar nature software, Frinika 0.2.0. This will allow us to externally validate the applicability of the identified threshold values. This will help in obtaining generalized and well-formed results.

We have used area under the receiver operating characteristic curve (AUC) and g-mean to analyse the results as these are suitable for unbalanced data. The results indicate that there exist thresholds for OO metrics and these thresholds can be applied by software engineers on similar nature software systems to find the change prone design and code areas that may require corrective action.

The paper is organized as follows: Next section, explains the basic background behind the work, emphasizing on the variables used and empirical data collection. Following this, in section 3, we present the research methodology that focuses on the basic approach or steps followed to carry out the work. It explains the univariate and the threshold results. Section 4 explains in detail the validation results of the study along with the results of external validation. Finally, the work is concluded in section 5.

2. RESEARCH BACKGROUND

In this section, we summarize the independent and dependent variables used in our study. We also explain the datasets used along with their descriptive statistics. The statistical method used and the formula for calculating threshold values have also been explained.

2.1 Dependent and Independent Variables

The dependent variable used in our study is change proneness. Change proneness can be defined as the probability of occurrence of change in a class. Change is calculated in terms of the number of lines of code added, deleted and modified in the recent version with respect to the previous version. The independent variables are a set of object oriented metrics listed below:

a. Coupling between object classes (CBO): Number of classes whose attributes is used by the given class plus those that use the attributes or methods of the given class
b. Number of Children (NOC): Number of direct children of a class in a hierarchy
c. Number of Attributes (NOA): Number of attributes/variables defined in a class
d. Number of Instance Variable (NIV): It measures relations of a class with other objects of the program
e. Depth of Inheritance Tree (DIT): Maximum number of steps from the class node to the root of the tree
f. Number of Methods per Class (NOM): Number of methods defined in the class
g. Number of Instance Method (NIM): Number of Instance Methods
h. Number of Local Methods (NLM): Number of local (not inherited) methods
i. Response For a Class (RFC): Number of methods in the class including the methods that are called by class's methods
j. Number of Local Default Visibility Methods (NLDM): Number of local default visibility methods
k. Number of Private Methods (NPRM): Number of local (not inherited) private methods
l. Number of Protected Methods (NPROM): Number of local protected methods
m. Number of Public Methods (NPM): Number of local (not inherited) public methods
n. Lack of Cohesion amongst methods (LCOM): For each data field in a class, the percentage of the methods in the class using that data field; the percentages are averaged and then subtracted from 100%

o. Weighted Methods Per Class (WMC): Count of sum of complexities of all methods in a class
p. Lines of Code (LOC): Number of lines that contain source code

2.2 Empirical Data Collection

The empirical validation is carried on an open source dataset, Freemind written in Java programming language. We analysed 3 releases of Freemind, 0.9.0, 0.9.1, 0.10.0. For external validation, another open source dataset is considered, Frinika written in Java programming language. The source code of these open source software is available at http://sourceforge.net. Freemind is a mind mapper and hierarchical editor. It serves various purposes to its users, including keeping track of projects, acts as workplace for Internet research, used for essay writing and brainstorming, etc. Frinika is a complete music workstation, which provides the end user with a complete platform for creating music with their computers. The details of software are provided in table 1. Change is collected for the common classes between 2 successive releases of each software. For example, change is collected in terms of number of lines added, deleted and modified in Freemind 0.9.0 with respect to Freemind 0.9.1. The common classes between Freemind 0.9.0 and 0.9.1 were found to be 656. Thus, we say that there are 656 datapoints for Freemind 0.9.0 where each datapoint corresponds to one Java class common in both the versions 0.9.0 and 0.9.1. For each data point, we collected: a) set of OO metrics for that class in first version, i.e. 0.9.0 in this case and b) the number of SLOC changes in that class from version 0.9.0 to 0.9.1. We carried out three analyses using the datasets as explained below (see Fig. 1):

1. We computed the threshold values of object oriented metrics on Freemind 0.9.0.
2. We validated the threshold values of the object oriented metrics on successive versions of Freemind (0.9.1, 1.0.0).
3. We validated the threshold values of the object oriented metrics obtained from Freemind 0.9.0 on Frinika 0.2.0. The external validation of Freemind is done on Frinika as they possess some common characteristics such as both are open source, free, written in java and licensed under GNU GPL. The purpose of this analysis was to compute the threshold values for the metrics and assess the applicability of the obtained threshold values on different datasets (including versions of the same software and different software).

Table 1: Details of Software

Software/version	Release year	KLOC	Total classes	Classes changed
Freemind 0.9.0	2008	48	656	29
Freemind 0.9.1	2009	50	608	118
Freemind 1.0.0	2011	68	668	555
Frinika 0.2.0	2009	50	248	126

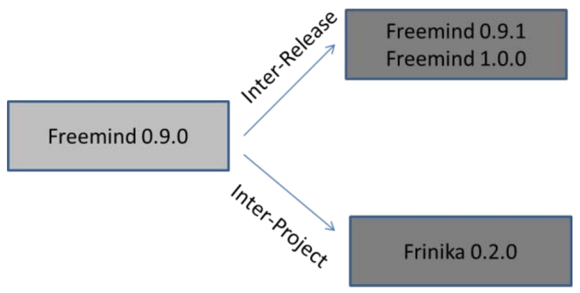

Fig. 1: Usage of datasets

3. RESEARCH METHODOLOGY

In this section, we briefly explain the methodology used. The first step is to conduct univariate analysis to find a subset of metrics which are significant predictors of change proneness. The next step is to calculate the threshold values for the significant metrics. The Bender method [9] based on the logistic regression (LR) method is used to calculate the thresholds at different levels of change (Po). We considered the threshold values at the lowest possible Po. Using these threshold values of the metric, we convert the metrics into binary. For validating the threshold values, we constructed various machine learning models. The results measured in terms of AUC and g-mean are compared with the results when machine learning models are applied on the non-binary original values (i.e. without thresholds).

3.1 Analysis of Univariate Logistic Regression

Logistic regression (LR) is the statistical method used to predict the dependent variable from a set of independent variables ([23], [24]). We have used univariate logistic regression to find a subset of significant metrics and to calculate the threshold values of those metrics. The univariate logistic regression formula is [24]:

$$P= e^{g(x)}/(1+e^{g(x)})$$

Where, P = probability of a class being change prone

x= independent variable

$g(x) = \alpha + \beta x$; α = constant and β = slope or estimated coefficient

The values of α and β are used to calculate threshold values. Table 2 shows the univariate results for Freemind 0.9.0. We calculate threshold values for the metrics of Freemind 0.9.0, thus univariate analysis is done only for Freemind 0.9.0. For each metric, the values of coefficient (β), constant (α), and statistical significance (sig.) are given. The ‘sig.’ parameter is used to indicate the association between each metric and change proneness. The ‘β’ parameter, known as coefficient shows the impact of the independent variable and its sign shows whether the impact is positive or negative. If the ‘sig.’ value is below or at the significance threshold of 0.05, then the metric is said to be significant to change proneness, else the metric is said to be insignificant. Only for significant metrics (shown in bold), we calculate the threshold values.

Table 2: Results of Univariate Analysis

Metrics	Constant (α)	Coefficient (β)	Sig.
CBO	-3.570	0.033	0.000
NOC	-3.048	-0.095	0.62
NOM	-3.159	0.061	0.002
NOA	-3.305	0.135	0.001
NIM	-3.352	0.026	0.001
NIV	-3.141	0.022	0.213
NLM	-3.457	0.031	0.000
RFC	-3.374	0.002	0.003
NLDM	-3.260	0.316	0.003
NPRM	-3.463	0.214	0.000
NPROM	-3.282	0.214	0.000
NPM	-3.332	0.029	0.001
LOC	-3.475	0.003	0.000
DIT	-3.674	0.225	0.056
LCOM	-3.925	0.018	0.001
WMC	-3.486	0.016	0.000

3.2 Threshold Analysis

We have used a statistical technique proposed by Bender [9] to calculate the threshold values. Value of an Acceptable Change Level (VACL) gives the threshold values for OO metrics. The formula for VACL is given as follows:

$$\text{VACL} = \frac{1}{\beta}\left[\log\left[\frac{\text{Po}}{1-\text{Po}}\right] - \alpha\right]$$

Where, α = constant

β = estimated coefficient

Po= acceptable change level

In this formula, α and β are obtained using the LR formula (explained in section 3.1). Po is defined as the acceptable change level, which is taken as a suggested probability (example, Po = 0.05 or Po = 0.01) by Bender [9]. For classes with metrics values below VACL, the probability of change occurrence is lower than Po. This variable (Po) can take different values, but according to its definition, it is clear that lower the value of Po, better it is.

For the significant metrics of Freemind 0.9.0, we calculate the threshold values. Table 3 shows the threshold values at different Po values, i.e. 0.01, 0.05, 0.08 and 0.1. The lowest value of Po where the threshold values are positive is considered and the threshold values at that Po are considered for validating Freemind 0.9.1, 1.0.0 and Frinika 0.2.0. We can observe that at Po values (0.05, 0.08 and 0.1); the VACL values of the metrics are within the range (i.e. positive values). Therefore, we will consider the threshold values at 0.05 for further analysis. We can observe from the table that VACL varies largely as the value of Po changes. For a small increase in the value of Po, VACL increases to a large extent. This shows that Po plays a significant role in calculating threshold values.

Table 3: Threshold Values of Freemind 0.9.0

Metrics	VACL at Po = 0.01	VACL at Po = 0.05	VACL at Po= 0.08	VACL at Po= 0.1
CBO	-31.064	18.956	34.171	41.599
NOM	-23.543	3.517	11.748	15.767
NOA	-9.556	2.671	6.39	8.206
NIM	-47.812	15.675	34.987	44.414
NLM	-36.714	16.534	32.731	40.638
RFC	-610.56	214.781	465.826	588.388
NLDM	-4.225	0.999	2.588	3.363
NPRM	-5.29	2.423	4.769	5.915
NPROM	-6.136	1.577	3.924	5.069
NPM	-43.556	13.364	30.678	39.13
LOC	-373.373	176.854	344.218	425.925
LCOM	-37.229	54.476	82.37	95.988
WMC	-69.32	33.848	65.228	80.548

4. RESULT ANALYSIS

In this section, we present the results of inter-release and inter-project validation. In other words, we have converted the metrics of Freemind 0.9.0, 0.9.1 and 1.0.0 to binary using the threshold values of Freemind 0.9.0. Similarly, the metrics of Frinika 0.2.0 are also converted to binary using threshold values of Freemind 0.9.0. After the metrics are converted to binary, correlation based feature selection (CFS) is applied to obtain the best subset of independent variables. Table 4 lists the metrics obtained after applying the CFS technique. These selected metrics are used to build a classification model using different machine learning techniques. We have used following two evaluation parameters to evaluate the results:

Area under Receiver Operating Characteristics Curve (ROC): ROC is a plot between sensitivity on the y-axis and (1-specificity) on the x-axis [25]. The area under the ROC curve (AUC) is a measure of accuracy of the predicted model. Higher the value of the AUC, higher is the accuracy of the model predicted.

G-mean (GM): It is defined as the square root of the product of (a+) and (a-) where (a+) and (a-) are known as the accuracy of positives and negatives respectively. The accuracy of positives (a+) is defined as the ratio of number of classes that are actually ‘change prone’ to the number of classes that are predicted to be ‘change prone’. The accuracy of negatives (a-) is defined as the ratio of number of classes that are actually ‘not change prone’ to the number of classes that are predicted to be ‘not change prone’. Mathematically, G-mean= square root ((a+)*(a-)).

Table 4: Metrics selected using CFS

Software	Metrics selected
Freemind 0.9.0	CBO, NLM, NPROM, WMC
Freemind 0.9.1	CBO, NPM
Freemind 1.0.0	NLDM, NPROM, NPM, LCOM, WMC
Frinika 0.2.0	CBO, NLDM, LOC, LCOM

4.1 Inter-Release Validation

Table 5 shows the results of validating different releases of Freemind. The results of validating same software from which thresholds are obtained (Freemind 0.9.0) show MLP to be the best classifier with the highest AUC and G-mean. Other classifiers have shown comparable performance. However, when performing inter-release validation, i.e. validating Freemind 0.9.1 and 1.0.0, the results show the best performance by CRT. For both the releases, AUC is above or equal to 0.7 and g-mean is also above 70%. Besides this, the results of inter-release validation are comparable to the results of validating Freemind 0.9.0. Thus, this shows the strength of our proposed methodology. In other words, thresholds derived for the metrics of previous release can be used to predict change prone classes in the future or upcoming release. MLP has shown the second best performance and other classifiers have shown comparable performance.

Table 5: Results of Inter-Release Validation

	Freemid 0.9.0		**Freemind 0.9.1**		**Freemind 1.0.0**	
Method	**AUC**	**GM**	**AUC**	**GM**	**AUC**	**GM**
AB	0.68	75.9	0.661	62.3	0.643	61.0
Bagging	0.58	51.7	0.675	63.4	0.575	54.0
LB	0.67	75.8	0.659	62.4	0.658	65.8
MLP	0.716	76.0	0.682	65.2	0.685	65.2
NB	0.686	75.4	0.665	64.0	0.662	65.8
RF	0.632	73.4	0.678	63.3	0.67	64.5
CRT	0.68	74.4	0.70	70.8	0.718	72.4

4.2 Inter-Project Validation

For evaluating the prediction accuracy of the model, inter-project validation is carried out on Frinika dataset. Table 6 presents the results for inter-project validation. We observe that the results of validating Frinika are almost similar for all the classifiers. Bagging, LB, MLP, NB and RF have shown exactly the same results for GM (64.8). Overall, we can say that we obtained competitive and consistent results for Frinika. We notice that no machine learning classifier actually outperformed, but all gave good and competitive results. This shows that the threshold values can be effectively utilized and applied on different datasets of similar nature.

Table 6: Results of Inter Project Validation

Dataset	**Frinika 0.2.0**	
Method	**AUC**	**GM %**
AB	0.637	63.7
Bagging	0.611	64.8
LB	0.621	64.8
MLP	0.628	64.8
NB	0.638	64.8
RF	0.607	64.8
CRT	0.612	64.8

5. CONCLUSION

The prediction of change prone classes in early phases of software development life cycle is gaining wide importance as it helps in improving the quality of software. It helps in efficient allocation of resources and thus, reduces the costs associated with the maintenance phase. Various object oriented (OO) metrics can be used for this purpose. In this paper, we have calculated the threshold values of various OO metrics which can be used to identify classes where focussed attention is required. Threshold values can be defined as certain alarming values above which a class is considered to be risky, In other words, the classes whose OO metrics exceed the threshold values (alarming values), can be selected for focussed attention and rigorous testing to improve the quality. We have sued a statistical based approach which uses logistic regression to calculate the threshold values. Threshold values are obtained for open source software Freemind 0.9.0 and validated on the same as well as different releases, Freemind 0.9.1 and 1.0.0. In addition to this, we also carried out inter-project validation where we take a different project, Frinika to validate the threshold values. All the models are validated using 10-cross validation and various machine learning classifiers are used such as adaboost, bagging, logitboost, multilayer perceptron, naïve bayes, random forest and classification & regression trees.

Following important conclusions can be made from this work:

1. Using univariate logistic regression on Freemind 0.9.0, we found that there is a significant relationship between the metrics and the dependent variable (change proneness). Majority of the metrics are found to be significant in predicting change proneness.
2. There are effective threshold values for the object oriented metrics and there is significant effect of Po values on the threshold values. Threshold values change as we change the value of change level (Po).
3. We calculated the threshold values at different values of Po (0.01, 0.05, 0.08 and 0.1). For Freemind 0.9.0, we found Po= 0.05 to be the most appropriate.
4. For validating the dataset on which thresholds are obtained (Freemind 0.9.0) along with different releases of the dataset (Freemind 0.9.1 and 1.0.0), we built various machine learning models. We found the performance of the models obtained for different releases (Freemind 0.9.1 and 1.0.0) is comparable to the performance of the models obtained on the release from which thresholds are obtained (Freemind 0.9.0). This shows our threshold methodology can be effectively utilized on upcoming or future releases of software. We also performed inter-project validation using Frinika dataset.

We plan to replicate our study on larger datasets so that the results can be made more generalized. Besides this, we also plan to predict models based on machine learning algorithms such as genetic algorithms. We would also like to do cost benefit analysis of the models predicted in our future studies.

REFERENCES

[1] Koru, A.G., and Tian, J., "Comparing High-Change Modules and Modules with the Highest Measurement Values in Two Large-Scale Open-Source Products," *IEEE Transactions on Software Engineering*, 31, 8, 2005, pp. 625–642.

[2] Koru, A.G. and Liu,H., "Identifying and characterizing change-prone classes in two large-scale open-source products," *The Journal of Systems and Software*, 80,1, 2007, pp. 63-73.

[3] Han, A.R., Jeon,S.U., Bae, D.H., and Hong,J.E., "Behavioural Dependency Measurement for Change-proneness Prediction in UML 2.0 Design Models," *Annual IEEE International Computer Software and Applications Conference*, 2008.

[4] Zhou,Y., Leung, H., and Xu,B., "Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness," *IEEE Transactions on Software Engineering*, 35, 5, 2009, pp. 607-623.

[5] Han, A.R., Jeon,S.U., Bae, D.H., and Hong,J.E., "Measuring behavioral dependency for improving change-proneness prediction in UML-based design models," *The Journal of Systems and Software*, vol. 83, 2010, pp. 222–234.

[6] Lu,H., Zhou,Y., Xu,B., Leung, H., and Chen,L., "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," *Empir Software Eng*., 17, 3, 2011, pp. 200-242.

[7] Malhotra, R. and Khanna,M., "Investigation of relationship between object-oriented metrics and change proneness", *Int. J. Mach. Learn. & Cyber*," 4, 4, 2013, pp. 273-286.

[8] Chidamber,S., Darcy, D., and Kemerer,C., "Managerial use of metrics for object-oriented software: an exploratory analysis," *IEEE Trans Softw Eng*., 24, 8,1998, pp. 629–639.

[9] Bender,R., "Quantitative Risk Assessment in Epidemiological Studies Investigating Threshold Effects," *Biometrical Journal*, 41, 3, 1999, pp. 305-319.

[10] McCabe,T.J., "A complexity measure," *Software Engineering, IEEE Transactions*, SE-2, 4, 1976, pp. 308-320.

[11] Coleman,D., Lowther,B., and Oman,P., "The application of software maintainability models in industrial software systems," *J. Syst. Softw.*, 29, 1, 1999, pp. 3-16.

[12] Erni, K., and Lewerentz,C., "Applying Design-Metrics to Object-Oriented Frameworks," *Proc. Third Int'l Software Metrics Symp*., Washington DC, USA, 1996, pp. 64-74.

[13] French,V.A., "Establishing software metric thresholds, "International Workshop on Software Measurement, 1999.

[14] Daly,J., Brooks,A., Miller,J., Roper, M., and Wood,M., "Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software," *Empirical Software Eng.*, 1, 2, 1996, pp. 109-132.

[15] Cartwright,M., "An Empirical View of Inheritance," *Information and Software Technology*, 40,14, 1998, pp. 795-799.

[16] Harrison,R., Counsell, S., and Nithi, R., "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems," *J. Systems and Software*, 52, (2/3), 2000, pp. 173-179.

[17] Prechelt,L., Unger,B., Philippsen, M., and Tichy,W., "A Controlled Experiment on Inheritance Depth as a Cost Factor for Code Maintenance," *J. Systems and Software*, 65,2, 2003, pp.115-126.

[18] El Emam,K. Benlarbi,S., Goel, N., and Rai,S., "Thresholds for Object-Oriented Measures," *Proc. 11th Int'l Symp. Software Reliability Eng.* 2000, pp. 24-38.

[19] El Emam,K., Benlarbi,S., Goel,N., Melo,W., Lounis H., and Rai, S., "The Optimal Class Size for Object-Oriented Software, "*IEEE Trans. Software Eng*., 28, 5, 2002, pp. 494-509.

[20] Shatnawi,R., "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems," *IEEE transactions on Software Engineering*, 36, 2, 2010, pp. 216 – 225.

[21] Benlarbi,S., El Emam,K., Goel,N., and Rai,S., "Thresholds for object-oriented measures," in *Proc. Of the 11th International Symposium on Software Reliability Engineering*, 2000, pp.24.

[22] Ulm,K., "A Statistical Method for Assessing a Threshold in Epidemiological Studies," *Statistics in Medicine*, 10, 3, 1991, pp. 341-349.

[23] Singh,Y., Kaur, A. and Malhotra,R., "Empirical validation of object-oriented metrics for predicting fault proneness," *Softw Qual J*., 18, 1, 2010, pp. 3–35.

[24] Hosmer D. and Lemeshow, S. (1989). Applied logistic regression (Wiley).

[25] El Emam,K., Benlarbi,S., Goel, N., and Rai,S., "A validation of object-oriented metrics," *NRC Tech. rep. ERB-1063*, 1999.