2008

An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction

Luiz Fernando Capretz
University of Western Ontario, lcapretz@uwo.ca

Jie Xu
Athos Origin China, jeffrey.xj@hotmail.com

Journal of Computer Science 4 (7): 571-577, 2008
ISSN 1549-3636

An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction

[1]Jie Xu, [2]Danny Ho and [1]Luiz Fernando Capretz
[1]Department of Electrical and Computer Engineering,
University of Western Ontario, London, Ontario, N6A 5B9 Canada
[2]NFA Estimation Inc., Oshawa, Ontario, Canada

Abstract: Problem Statement: Object-oriented design has become a dominant method in software industry and many design metrics of object-oriented programs have been proposed for quality prediction, but there is no well-accepted statement on how significant those metrics are. In this study, empirical analysis is carried out to validate object-oriented design metrics for defects estimation. **Approach:** The Chidamber and Kemerer metrics suite is adopted to estimate the number of defects in the programs, which are extracted from a public NASA data set. The techniques involved are statistical analysis and neuro-fuzzy approach. **Results:** The results indicate that SLOC, WMC, CBO and RFC are reliable metrics for defect estimation. Overall, SLOC imposes most significant impact on the number of defects. **Conclusions/Recommendations:** The design metrics are closely related to the number of defects in OO classes, but we can not jump to a conclusion by using one analysis technique. We recommend using neuro-fuzzy approach together with statistical techniques to reveal the relationship between metrics and dependent variables, and the correlations among those metrics also have to be considered.

Key words: Software quality, design metrics, statistical analysis, neuro-fuzzy, prediction

INTRODUCTION

Time, cost and scope are regarded as the three pillars of software project management whereas quality is applicable to all. It is a well-proved fact that the earlier a defect is found and fixed, the less it costs. Thus high quality products are plausible to be delivered. Unfortunately, as the size and complexity of software grows dramatically in modern industry, the quality becomes very hard to predict or estimate.

Researchers and engineers have been working on this subject for more than three decades, and many started with static design metrics in the programs. Those metrics of complexity, such as Source Lines of Code (SLOC)[1], Halstead's software science[2] and McCabe's cyclomatic complexity[3], all impose their influence on software quality at module level. Nonetheless, the relationships between those metrics and software quality are not accurately elaborated, and hardly can satisfactory estimation results be achieved.

After object-oriented programming dominated software development, a vast variety of design metrics have been tailored for estimating the quality of object-oriented programs.

Chidamber and Kemerer[4] introduced their OO design and complexity metrics and demonstrated the strong impact on software quality. The CK metrics suite invoked great enthusiasm among researchers and software engineers, and a great amount of empirical studies have been conducted to evaluate those metrics. Moreover, other variants of CK metrics were added in order to present more accurate indications of code quality. Although all these studies made valuable contributions to improve OO design, their results seemed not consistent[5-15].

In this study, data from the industry is used to analyze the relationships between CK metrics and defects in the OO programs. Besides statistical analysis methods, we utilize a neuro-fuzzy approach to validate the relationships.

CK metrics: In CK metrics suite[4], six design and complexity metrics are used to represent the characteristics of the code:

- WMC (Weighted Methods per Class): The sum of normalized complexity of every method in a given class. Usually we just use the number of methods in a given class

Corresponding Author: Jie Xu, Department of Electrical and Computer Engineering, University of Western Ontario, London, Ontario, N6A 5B9 Canada

Table 1: Descriptive information of metrics

Metric	Minimum	Maximum	Median	Mean
CBO	0	24	8	8.3
DIT	1	7	2	2.0
LCOM	0	100	84	68.7
NOC	0	5	0	0.2
RFC	0	222	28	34.4
WMC	0	100	12	17.4
SLOC	0	2313	108	211.2
Defects	0	101	0	4.6

- DIT (Depth of Inheritance Tree): The maximum length from the root to a given class in the inheritance hierarchy
- NOC (Number Of Children): The number of all direct subclasses of a given class
- RFC (Response For a Class): The number of methods implemented in a given class that can be invoked by a received message
- CBO (Coupling Between Object Classes): the number of classes that use the member functions and/or the instance variables of a given class
- LCOM (Lack of Cohesion on Methods): for each instance variable calculate the percentage of methods using it, then the average percentage for all variables subtracted from 100%
- Intuitively, these CK metrics illustrate the measurable characteristics of an OO program, such as complexity, cohesion and coupling. For general design environment, we technically should keep all those metrics at a reasonable level

Data preparation: In this study we use a data set KC1 from the NASA IV and V Facility Metrics Data Program data repository (http://mdp.ivv.nasa.org), which is comprised of 43 KSLOC of C++ code for a ground system. There are 145 classes and altogether 2107 methods.

The defects information in the original data set is at method level, while metrics information is at class level. For this study, those files have to be combined to get all class level information. We generate all class level information and validate it with the above data resource.

Descriptive information about this public data set is listed in Table 1. Since six CK metrics and Source Lines of Code (SLOC) are examined to evaluate their impact on the quality of the code, only related information is included in the Table 1.

The intention of this study is to validate and compare the influence of those metrics on the final quality of software, so transformation should be made to maintain those metrics at comparable levels. In this study, standardization of the metrics is pre-processed to make their means zero and standard deviation one.

MATERIALS AND METHODS

Correlation analysis: First, the relationships between each explanatory variable (metric) and the dependent variable (the number of defects) are revealed by using Spearman's rank correlation coefficient method. Although correlation between two variables does not necessarily result in causal effect, it is still an effective method to choose candidate metrics.

Regression analysis: Linear regression is the most popular statistical method to establish the relationship model between the explanatory variables and the dependent variable. Generally Ordinary Least Square (OLS) tries to obtain the least square estimates to build the regression model. When the intrinsic relationship is not linear, the regression can still be used to evaluate the influences of different predictors.

To explore the impact of various explanatory variables (metrics) on the dependent variable (defects), both university and multivariate linear regression are applied to the standardized metrics and the number of the defects.

In this research, the purpose of the linear regression is not to calibrate the accuracy of the estimation model, but to validate those object-oriented design metrics. So we just ignore the assumptions leading to linear regression.

In many related researches, logistic regression was extensively used to examine the relationships between object-oriented design metrics and the probability of the class containing faults, not the actual number of the faults. This regression analysis, on the other side, also exhibits the contribution of the metrics to the quality of the programs.

Neuro-fuzzy approach: From this study and Zhou and Leung's[15], obviously neither linear regression nor logistic regression can achieve a model to elaborate the complexity between those metrics and the quality of code. Because of the diversity of software engineering practices, no simple equation can be discovered to delicately abstract the relationship from industry data.

Since neural networks began to regain public interests in 1982, they have been widely implemented in a variety of fields as engineering, business, medicine and physics. Their performance in prediction, classification, and real time control is highly recognized by experts in those domains.

The main strength of neural networks is their power to model extremely complex functions, for example, for the cases where there are a great number of variables and traditional regression methods are not applicable. Moreover, neural networks can learn from historical data and train themselves to achieve high performance, where not much expertise is mandatory.

Another sophisticated technique for modelling complex systems is fuzzy logic, which is applied with heuristic knowledge and with imprecise inputs to realize complicated functions. Because the real world is full of vagueness, fuzzy logic has proved to be very successful in many fields, like dynamic control, decision support, and other expert systems.

Both neural network and fuzzy logic have their advantages in modelling nonlinear functions, but their strengths attract us from quite different aspects. Moreover, since neural network lacks the capability of explaining the meaning behind the application, which is exactly the strength of fuzzy logic, it has drawn great attention to combine the two techniques together to form a new approach. Here comes neuro-fuzzy, which is now an important constituent of soft computing, and is applied to solve decision-making, modelling, predicting, and control problems in the real world.

In this study, we adopt ANFIS (Adaptive Neuro-Fuzzy Inference System) to approximate the function between design metrics and defects. ANFIS is actually a fuzzy inference system, which is based on a neural network structure, thus it has the learning capability inherited from neural networks to adjust the parameters of membership functions and rules. A typical structure of ANFIS is shown in Fig. 1.

ANFIS has proved to have extraordinary power for realizing arbitrary nonlinear functions[16]. Therefore, it is a quite suitable tool for evaluating the relationship between design metrics and defects. ANFIS is comprised of five layers:

Layer 1: Every node in this layer is an adaptive node. Parameters in this layer are called premise parameters.

Layer 2: Every node in this layer is a fixed node labelled $\prod$, whose output is the product of all the incoming signals. Each node output represents the firing strength of a rule.

Layer 3: Every node in this layer is a fixed node labelled N. The ith node calculates the ratio of the ith rule's firing strength. Thus the outputs of this layer are called normalized firing strengths.

Layer 4: Every node i in this layer is an adaptive node. Parameters in this layer are referred to as consequent parameters.

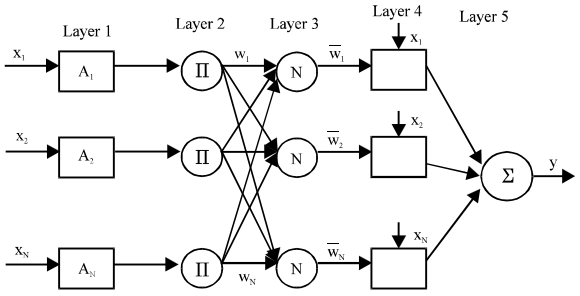

Fig. 1: The typical structure of ANFIS

Layer 5: The single node in this layer is a fixed node labelled $\sum$, which computes the overall output as the summation of all incoming signals.

In this study, only 4 metrics, CBO, RFC, WMC and SLOC, which show their tight correlation with defects numbers, are chosen as input variables in ANFIS. Because these four are highly correlated with each other, we cannot simplify the rule set when we design the structure of ANFIS. Therefore there are altogether 81 (3 powered by 4) rules if each metric has 3 membership functions. Even though we choose constants as the consequent parts of the fuzzy rules and triangular membership functions for the purpose of simplicity, totally 117 parameters need to be adjusted during the whole training procedure. As regard to the NASA data set, which includes only 145 data points, it is apparently not enough to gain satisfactory training result. For this reason, together with easy explanation, we classify the metrics only into two levels, 'high' and 'low'. Still keeping present constant and triangle choice, we diminish the number of parameters to 40 (16 rules, 8 functions). Every rule has the form as:

If CBO is high AND RFC is high AND WMC is high AND SLOC is high, then the number of defects is N1.

We precede present experiments with Matlab, which has built-in ANFIS function and editor that make the implementation and customization more convenient for us. The structure of present ANFIS is shown in Fig. 2.

To initialize and train the ANFIS, we divide the data set into training data (120 data points) and checking data (25 data points), and the data points randomly fall into one group. We do not expect to obtain a well-tuned ANFIS model due to the insufficiency of data. The main purpose is to retrieve straightforward information out of the built system.

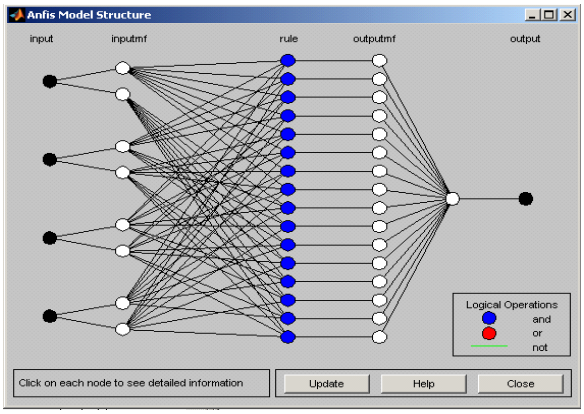

Fig. 2: The structure of ANFIS in Matlab

RESULTS

Correlation analysis: Table 2 displays the results, which shows that CBO, RFC, WMC and SLOC have more influence on the number of defects in the programs (significance at 0.01 levels). Among them, SLOC seems to be the most important indicator of defects, which has the biggest coefficient.

We also examine the Spearman's rank correlation among the seven metrics (6 CK metrics plus SLOC). The results (Table 3) illustrate these metrics are far from being independent, but correlated with each other, especially there are strong correlations among CBO, RFC, WMC and SLOC.

Regression analysis: The result of the univariate linear regression is listed in Table 4, which shows SLOC, WMC, CBO and RFC are more significant with respect to other metrics for their higher coefficients and R-square values at significant levels. The SLOC is regarded as the most effective one, since it captures around 45 percent of the variance.

The result of the multivariate linear regression is also displayed in Table 5. This time SLOC, CBO, RFC and WMC still come out to be stronger than other metrics, which demonstrate some unstable impact on the dependent variable (the coefficients are 7.3408, 0.8775, 0.6662 and 0.3329 accordingly). Moreover, the R-square value is 0.5098 for the multivariate linear regression, which shows that nearly half of the variance is not included in this model. Again, we will not strive to exploit how to improve the accuracy of the model, but the multicollinearity among the metrics and the heteroscedasticity of the errors definitely contribute to the deviation. The result of another linear regression using only four metrics is listed in Table 6.

Table 2: Correlations between metrics and defects

Metrics	Coefficients	p-value
CBO	0.5472	0.0000
DIT	0.0411	0.6232
LCOM	-0.0178	0.8319
NOC	-0.1743	0.0360
RFC	0.2475	0.0027
WMC	0.3383	0.0000
SLOC	0.5763	0.0000

Table 3: Inter-correlations among the metrics

	DIT	LCOM	NOC	RFC	WMC	SLOC
CBO	0.4703*	0.0413	-0.1287	0.5416*	0.3773*	0.8184*
DIT		0.2495	-0.1123	0.7179*	0.2154	0.3853*
LCOM			0.1193	0.3827*	0.3780*	0.0097
NOC				-0.0317	0.1014	-0.1363
RFC					0.6661*	0.5230*
WMC						0.4963*

"*" indicates at the 0.001 significance level

Table 4: Univariate linear regression

	CBO	DIT	LCOM	NOC	RFC	WMC	SLOC
Coefficient	3.9704	0.4029	1.3695	-1.1427	2.9552	5.0308	7.3106
Constant	4.6138	4.6138	4.6138	4.6138	4.6138	4.6138	4.6138
R. square	0.1337	0.0014	0.0159	0.0111	0.0741	0.2147	0.4533
P-value	0.0000	0.6577	0.1306	0.2078	0.0009	0.0000	0.0000

Table 5: Multivariate linear regression

	Constant	CBO	DIT	LCOM	NOC	RFC	WMC	SLOC
Coefficient	4.6138	0.8775	-2.9951	-0.1842	-0.9080	0.6662	0.3329	7.3048

Table 6: Multivariate linear regression (2)

	Const.	CBO	RFC	WMC	SLOC
Coef.	4.6138	0.2125	-1.7731	1.7042	7.0207

Table 7: Multivariate logistic regression

	Const.	CBO	RFC	WMC	SLOC
Coef.	1.8045	1.3274	-0.6817	0.2111	1.8925
P	0.0000	0.0000	0.0006	0.2620	0.0023

Table 8: Fuzzy fules from trained anfis

	CBO	RFC	WMC	SLOC	Output
1	Low	Low	Low	Low	2.320
2	Low	Low	Low	High	-4.195
3	Low	Low	High	Low	-10.270
4	Low	Low	High	High	226.500
5	Low	High	Low	Low	-3.607
6	Low	High	Low	High	-50.070
7	Low	High	High	Low	16.450
8	Low	High	High	High	-35.130
9	High	Low	Low	Low	-11.160
10	High	Low	Low	High	-258.300
11	High	Low	High	Low	272.400
12	High	Low	High	High	263.200
13	High	High	Low	Low	45.070
14	High	High	Low	High	810.500
15	High	High	High	Low	-754.500
16	High	High	High	High	-420.400

Table 9: Aggregated outputs with different inputs

	CBO	RFC	WMC	SLOC	Output
1	0.0000	0.0000	0.0000	0.0000	2.570
2	2.4594	0.0000	0.0000	0.0000	12.100
3	0.0000	5.1825	0.0000	0.0000	4.730
4	0.0000	0.0000	4.7326	0.0000	13.700
5	0.0000	0.0000	0.0000	6.0832	25.000
6	-1.3043	0.0000	0.0000	0.0000	2.570
7	0.0000	-0.9496	0.0000	0.0000	3.810
8	0.0000	0.0000	-0.9984	0.0000	0.626
9	0.0000	0.0000	0.0000	-0.6113	0.563

We have observed that[15] built a logistic regression model based on the same NASA data set, and both university and multivariate logistic regression were applied to identify the effectiveness of the metrics. Actually we get the similar university and multivariate regression results as in their study, so we do not contain those logistic regression results in the study. Instead, only a result of regression using several metrics is shown in Table 7.

Neuro-fuzzy approach: We list all the fuzzy rules after training in Table 8. Contrary to present expectation, the results are not entirely consistent and we cannot elicit valuable information out of them. The reasons behind the unsuccessful training are: 1) the data set does not contain profound information to cover possible distribution of the metrics and defects; 2) for the trained membership functions, the 'high' and 'low' definitions do not agree with the standardized metrics values. For example, the range of WMC is [-0.9984, 4.7326] and the mean value is 0, but the membership function cannot follow it because of the unbalance of the data.

To reveal the output trend with the standardized values of the explanatory variables, we choose one input to set it to high or low end, while keeping other inputs at mean value 0, to examine the change of the aggregated output. The results are shown in Table 9. Obviously, SLOC influences the output most, then come WMC and CBO, but RFC seems to impose some impact on the output from the negative direction, which does not totally comply with the statistical analysis, and we will discuss about it later.

We also perform ANFIS with each explanatory variable separately to see whether it is able to achieve good prediction independently. To set up ANFIS structure, we employ k-means clustering to categorize each variable. The results, especially the root mean squares, show that SLOC is more effective than the others, because its result of the root mean square is around 5 while the others are above 10.

DISCUSSION

Related works: Most related published study focused on validating the effectiveness of object-oriented design metrics for predicting fault-prone classes, instead of the numbers of defects/faults in the lasses. Nevertheless, we can consider them for comparison; especially those investigating the same CK metrics.

Basili *et al*.[5] collected data on eight medium-sized systems that were designed to meet the same requirement specification and implemented with C++. The metrics under investigation were the CK metrics suite and logistic regression was utilized to perform the data analysis. They claimed that most CK metrics (except LCOM) were effective predictors for class fault-proneness, and among them, DIT and RFC were believed to have more influence on the dependent variable. They also concluded that there was hardly any correlation among the metrics.

Tang *et al*.[7] conducted their study on data from an industrial system, which was comprised of more than 200 C++ subsystems under Windows NT. Some metrics other than CK metrics were added to the metrics group under investigation, and logistic regression was carried out to evaluate those metrics. The results illustrated that WMC and DIT were significant indicators for finding fault-prone classes, but still, CK metrics seemed not sufficient.

Emam *et al*.[9] also chose logistic regression to analyze data from a telecommunication system, which consisted of 174 C++ classes. They found WMC, RFC and CBO were closely associated with fault-proneness; on the other hand, when size controlling was imposed on the analysis, the significance of the metrics no longer existed. They urged other researchers to re-examine their study with size controlling.

Yu *et al*.[11] completed another validation study of CK metrics with data from the client side application of a large network service management system, which contained 123 Java classes and approximate 34,000 lines of code. The dependent variable once again was the fault-proneness of the classes, and linear regression and discriminant analysis were their analysis methodology. They came to the conclusion that most CK metrics (except DIT) were sound predictor for the fault-prone classes.

Subramanyan *et al*.[12] accomplished their study based on the data collected from a large B2C e-commerce system, which was developed in C++ and Java to work on AIX and Windows NT. They only examined metric data of WMC, CBO and DIT from 405 C++ classes and 301 Java classes and used linear regression with Box-Cox transformation to determine the relationships between metrics and defects.

Table 10: Summary of related works

Study	Method	Lang.	Dependent Variable	CBO	DIT	LCOM	NOC	RFC	WMC	SLOC
Basili *et al*.[5]	LR	C++	Fault-proneness	+	++		−−	++	+	
Tang *et al*.[7]	LR	C++	Fault-proneness	0	0		0	+	+	
Emam *et al*.[9]	LR	C++	Fault-proneness	+	0	0		+	+	
Yu *et al*.[11]	OLS+LDA	Java	Faults	+	0	+	+	+	++	
Subramanyan *et al*.[12]	OLS	C++	Faults	+	−				+	+
		Java		−	−				0	++
Succi *et al*.[13]	PRM, NBRM + ZINBRM	C++	Faults		++			++		
Gyimóthy *et al*.[14]	LR+ML	C++	Fault-proneness	++	0	++		++	++	++
Zhou *et al*.[15]	LR+ML	C++	Fault-proneness	++	0	+	−−	++	++	++
Our study	OLS+ANF IS	C++	Faults	++				−	++	+++

LR: Logistic Regression; OLS: Ordinary Least Square; LDA: Linear Discriminant Analysis; PRM: Poisson Regression Model; NBRM: Negative Binominal Regression Model; ZINBRM: Zeros-Inflated Negative Binominal Regression Model; ML: Machine Learning; ANFIS: Adaptive Neuro-Fuzzy Inference System

Interestingly, the results of the impact of the metrics were not consistent between two programming languages, although size was always an effective indicator.

Succi *et al*.[13] assessed data from two commercial applications implemented using C++. One had 150 classes and 23 KSLOC (thousands of SLOC), while the other 144 classes and 25 KSLOC. The six CK metrics were under assessment to evaluate their influence on the number of defects. With Poisson, NB and zero-inflated regression analysis, they found out that RFC and DIT were the most valuable explanatory variables.

Gyimóthy *et al*.[14] found open source software was a great source for their validation of object-oriented metrics. Their samples were from Mozilla, and 3,192 C++ classes were inspected. Analysis methodology including logistic regression, linear regression and machine learning proceeded to validate SLOC and CK metrics on class fault-proneness. The results showed all CK metrics excluding NOC and SLOC were powerful explanatory variable. Zhou *et al*.[15] also explored NASA KC1 data set as in this study, and took severity levels of defects into account when researching the relationships between CK metrics and fault-prone classes. Their analysis methods were logistic regression and machine learning. They maintained that CBO, WMC, RFC and LCOM were significant while DIT was not significant regardless of severity levels. Moreover, they held that severity levels could greatly influence the predicting power of CK metrics upon class fault-proneness.

A summary of related research was listed in Table 10, and we can easily find: 1) the fault-proneness was the most employed dependent variable in those studies, since 5 out of 8 used it. 2) CBO, WMC and RFC were widely accepted as useful indictors for faults or fault-prone classes (5 out of 8 drew similar conclusions).

CONCLUSION

In this study, we analyzed KC1 data set from the public NASA repository to validate and clarify the significance of CK metrics as indictors for faults/defects in the programs. Although most previous studies tried to reveal their effects on the fault-proneness of the classes, we believe metrics and models that can predict the number of faults/defects in a class should be more valuable.

We utilized statistical methods and neuro-fuzzy approach to evaluate the relationships between CK metrics (together with SLOC) and faults/defects in the classes. The results could be summarized as:

- CK metrics do not demonstrate as powerful impact as SLOC. SLOC should continue to be a useful metric for object-oriented programs
- Among CK metrics, WMC, CBO and RFC are qualified indicators for predicting faults in classes and other metrics are trivial, while RFC demonstrates more complicated effect on the number of defects in classes. We should not determine the influence of one single factor separately, instead, we need to put related factors into one model
- Up to date, no superior model or algorithm can do accurate prediction across projects in software quality assurance
- Since the data we analyzed in this study is from one single project and it is not sufficient to build an accurate model for defects estimation, our next step will be collecting data from industrial projects. By using customized neuro-fuzzy approach, we will not only validate design metrics like CK metrics suite, but also investigate other process and personnel metrics

REFERENCES

1. Basili, V.R. and B.T. Perricone, 1984. Software errors and complexity: An empirical investigation. Commun. ACM, 27: 42-52. http://portal.acm.org/citation.cfm?id=2085.
2. Halstead, M.H., 1977. Elements of Software Science. 1st Edn., Elsevier North Holland, New York, ISBN: 10: 0444002057 pp: 127.
3. McCabe, T.J., 1976. A complexity measure. IEEE Trans. Software Eng., 2: 308-320. DOI: 10.1109/TSE.1976.233837.
4. Chidamber, S.R. and C.F. Kemerer, 1994. A metrics suite for object-oriented design. IEEE Trans. Software Eng., 20: 476-493. DOI: 10.1109/32.295895.
5. Basili, V.R., L.C. Briand and W.L. Melo, 1996. A validation of object-oriented design metrics as quality indicators. IEEE Trans. Software Eng., 22: 751-761. DOI: 10.1109/32.544352.
6. Briand, L.C., J. Wust, S.V. Ikonomovski and H. Lounis, 1999. Investigating quality factors in object-oriented designs: An industrial case study. Proceedings of the 21st International Conference on Software Engineering, May 16-22, ACM Los Angeles, California, United States, New York, USA., pp: 345-354. http://portal.acm.org/citation.cfm?id=302405.302654&type=series.
7. Tang, M.H., M.H. Kao and M.H. Chen, 1999. An empirical study on object-oriented metrics. Proceedings of the 6th International Symposium on Software Metrics, Oct. 04-06, IEEE Computer Society, Boca Raton, FL., USA., pp: 242-249. DOI: 10.1109/METRIC.1999.809745.
8. Briand, L.C., J. Wust, J.W. Daly and D.V. Porter, 2000. Exploring the relationships between design measures and software quality in object-oriented systems. J. Syst. Software, 51: 245-273. http://resolver.scholarsportal7.info/resolve/01641212/v51i0003/245_etrbdmasqios&form=pdf&file=file.pdf.
9. El Emam, K., S. Benlarbi, N. Goel and S.N. Rai, 2001. The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans. Software Eng., 27: 630-650. DOI: 10.1109/32.935855.
10. Briand, L.C., J. Wust and H. Lounis, 2001. Replicated case studies for investigating quality factors in object-oriented designs. Empirical Software Eng., 6: 11-58. http://resolver.scholarsportal.info/resolve/13823256/v06i0001/11_rcsfiqfiod&form=pdf&file=file.pdf.
11. Yu, P., T. Systa and H. Muller, 2002. Predicting fault-proneness using oo metrics: An industrial case study. Proceedings of the 6th European Conference on Software Maintenance and Reengineering, Mar. 11-13, Budapest, Hungary, pp: 99-107. DOI: 10.1109/CSMR.2002.995794.
12. Subramanyan, R. and M.S. Krisnan, 2003. Empirical analysis of CK metrics for object-oriented design complexity. Implications for software defects. IEEE Trans. Software Eng., 27: 297-310. http://www2.computer.org/portal/web/csdl/doi/10.1109/TSE.2003.1191795.
13. Succi, G., W. Pedrycz, M. Stefanovic and J. Miller, 2003. Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics. J. Syst. Software, 65: 1-12. DOI: 10.1016/S0164-1212(02)00024-9.
14. Gyimóthy, T., R. Ferenc and I. Siket, 2005. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans. Software Eng., 31: 897-910. DOI: 10.1109/TSE.2005.112.
15. Zhou, Y. and H. Leung, 2006. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Trans. Software Eng., 32: 771-789. DOI: 10.1109/TSE.2006.102.
16. Jang, J.S.R. C.T. Sun and E. Mizutani, 1997. Neuro-Fuzzy and Soft Computing. 1st Edn., Prentice Hall, New Jersey, ISBN: 10: 0132610663, pp: 335-345.