Information Technology Journal 7 (4): 698-701, 2008
ISSN 1812-5638
© 2008 Asian Network for Scientific Information

Fault Proneness Model for Object-Oriented Software: Design Phase Perspective

[1]R.A. Khan and [2]K. Mustafa
[1]Department of Information Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, UP, India
[2]Department of Computer Science, Jamia Millia Islamia, N. Delhi, India

Abstract: This research proposes a new method to estimate fault proneness of a class in design phase. A fault proneness model for object-oriented software (^{d}FPM) has been proposed as a major contribution of this study. The proposed model may be used in early stage of software development life cycle to predict the fault proneness of object-oriented software. Another effort has been made to quantify relevant metric suit. Proposed model has been validated with the help of hypothesis testing.

Key words: Software fault proneness, object oriented design metrics, logistic regression, object oriented software characteristics, regression model, Chi-square test

INTRODUCTION

To release a zero defect product is the dream of every developer. In order to achieve defect less product, companies spend 50 to 80% of their software development effort on testing (Collofello and Woodfield, 1989). Therefore reducing testing effort may increase productivity, reduce cost and optimize resources. Software design is the backbone of software development life cycle. Identification of faulty-modules in design phase reveals an effective and efficient test plan execution.

Software metrics are the measurement tools to be used to assess software products or related process. Numerous software metrics have been proposed in literature for software fault proneness. Software metrics have been developed for evaluating and quantifying several aspects of software engineering process (Fenton and Pfleeger, 1997; Gill and Kemerer, 1991). It has been revealed from the literature survey that metrics can be designed to localize fault-prone modules. Several studies have been done by researchers and practitioners to validate the existence of correlation between fault proneness and metrics (Basili and Hutchen, 1989; Gill and Kemerer, 1991; Li and Cheung, 1987; Silby and Basility, 1991; Denaro *et al*., 2002; Kanmani *et al*., 2006; Reddy *et al*., 2007; Rothermel *et al*., 2000, 2002).

Object Oriented Software differs from structured software in terms of its abstraction and real world modeling concepts that take the form of object oriented design constructs. A fundamental constraint of object oriented modeling and design is the Object, which combines both data structure and behavior in a single entity. Object-oriented technology provide product with higher quality and lower cost (Khan and Mustafa, 2004). Most of the available object oriented metrics can be used in code phase and hence late information for improvement (Khan and Mustafa, 2004). Early availability of object-oriented metrics may lead early information regarding faulty modules. Fault proneness models can be built using different techniques including machine learning principle (Brind *et al*., 2004), probabilistic approach (Rumbaugh *et al*., 2001), statistical techniques (Victor *et al*., 1996) and mixed techniques (Fenton and Neil, 1999). This study identifies and discusses four object oriented metrics to be used to calculate fault proneness of software. A fault proneness model has been proposed and validated at last.

OBJECT-ORIENTED CONCEPT

Object-oriented software provides an effective framework to analyze, design and prototype systems. Four fundamental characteristics of object-oriented approach are Encaplution, Inheritances, Coupling and Cohesion. Encapsulation is also known as information hiding in which data and some operations are hidden and inaccessible (Rumbaugh *et al*., 2001). Inheritance is the form the reuse that allows a programmer to define objects incrementally by reusing previously defines objects as the basis for new objects (Fenton and Neil, 1999). Coupling can be defined as degree of interconnectivity, joining and linking of entities. Cohesion is a high degree of internal relatedness of elements (Rumbaugh *et al*., 2001; Elrad *et al*., 2001).

Corresponding Author: R.A. Khan, Department of Information Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, UP, India

The following experimental hypothesis (Fenton and Neil, 1999) shows the relationship of above characteristics with fault-proneness.

Encapsulation: Class with more member functions is more complex and tends to be more fault-prone.

Inheritance: A class located deeper in a class inheritance lattice is supposed to be more fault-prone because class inherits as larger number of definitions from ancestors.

Coupling: Highly-coupled classes are more fault-prone than weekly-coupled classes because they depend on methods and objects defined in other classes.

Cohesion: Class with low cohesion among its methods suggests an unappropriate design, which is likely to be more fault-prone.

OBJECT-ORIENTED DESIGN METRICS

Metric is the unit of measurement of software attributes like size, cost, time required, complexity etc. A lot of time and resource are required for the development of large software systems. So accurate planning and proper allocation of resources is mandatory for different software activities. Software metrics are necessary to identity where the resource is needed.

Therefore, it is required to define and validate the metrics specific for object-oriented paradigm. To address this issue, several object-oriented metrics have recently been proposed by Kamia *et al.* (2005), Witten and Frank (2000), Fenton and Neil (1999), Baroni *et al.* (2002), Abreu (2001) and Stojanovic and Emam (2001). Among them Chidamber and Kemerer's metrics are well known ones as object-oriented metric suit. The effectiveness of these metrics has been empirically evaluated from viewpoint of software fault proneness. In the evaluation these metrics were applied to the source code because some of them measure inner complexity of a class. This study uses a metrics suite, which could be applied to design specification (Bansia and Devis, 2002).

Objects-oriented design metrics identification: Various object-oriented metrics have been proposed in the literature. Most of metrics lack validation. Exhausted review of literature on object oriented design metrics reveal four metrics ENM, INH, CPM and COM to be used in early stage of software development life cycle (Bansia and Devis, 2002). These metrics are discussed in Table 1. Table 2 shows the object oriented design constructs and relevant metrics to be used to quantify the software characteristics.

Table 1: Identified OO metrics

OO metric	Definition
ENM	No. of operations in a class
INM	Depth of inheritance in a class
CPM	No. of classes to which target class is coupled
COM	Count of relations among methods of a class

Table 2: Identified OO design constructs and relevant metrics

Design property	Definition	Metrics
Encapsulation	A kind of abstraction that enforces a clean separation between the external interface of an object and its internal implementation.	ENM
Inheritance	A measure of the is-a relationship between classes.	INH
Coupling	Interdependency of an object on other objects in a design.	CPM
Cohesion	High degree of internal relatedness of elements	COM

SOFTWARE FAULT PRONENESS

Software fault proneness is a key factor for monitoring and controlling the quality of software. The effectiveness of analysis and testing activities can be easily judged by comparing predicted distribution of fault (fault proneness) and amount of fault found with testing (software faultiness) (Denaro and Pezze, 2003).

Fault proneness of a class predicts the probability of the presence of faults in that class. Software testing and analysis are complex and expensive activities (Denaro and Pezze, 2002). Estimating and preventing the faults early and accurately is the better approach for reducing the testing effort. If fault prone modules are known in advance, review, analysis and testing efforts can be concentrated on those modules (Munson and Khoshgoftar, 1992).

Logistic regression: Fault proneness models can be built using many different methods including decision tree, neural network, Bayesian belief network, optimization set reduction and logistic regression. In this study logistic regression has been used to build fault proneness model based on historical data. The logistic regression model predicts the probability of software modules to be faulty of non-faulty based on values of metrics calculated on the modules (Denaro and Pezze, 2002). The variables describing the classes to be estimated is called dependent available of model. The variables that quantify the object attributes are called independent variable of model (Denaro and Pezze, 2002).

The Multivariate logistic regression model has been discussed in Eq. 1.

$$P = 1/ \{(1+\exp(-(C_0+C_1X_1+\text{----------}+CnXn))\} \quad (1)$$

Where:

P = Probability of error

X_i = Independent variables

P = Dependent variable

Ci = (Regression Co-efficient) is average amount of dependent increase when the independents are held constant (Victor *et al*., 1996)

A Fault proneness model has been developed using the identified metric suit and is shown in Eq. 2.

$$P = {}^{d}FPM = 1/1+e^{-(C1ENM+C2INM+C3COM+C4CPM)} \quad (2)$$

If the value of P is greater than 0.5, class is predicted to have faults (Kamia *et al*., 2005).

EXPERIMENTAL VALIDATION

This research gives the overview of assessment criteria used for validating the proposed model. The goal was to investigate whether the proposed model is capable of predicting faulty classes of a software project in design phase. Validation of the predictability of ^{d}FPM requires the similar set of object oriented software project. The faulty and non-faulty classes were predicted using ^{d}FPM. Faulty data was generated and actual faulty and non-faulty classes were detected using traditional approach. A medium size C++ project was selected for validation exercise. Metric values (ENM, INM, CPM, COM) of each class have been calculated along with the coefficient using High-Level Diagram (HLD).

Table 3 shows the distribution of the identified object oriented metric suit based 11 classes. Table 4 shows model based fault prediction and experimental results with comparative analysis.

Analysis and interpretation: Examining Table 4 shows that all of the metrics are highly correlated with each other. In order to further assure, χ^2 test has been used for testing the null hypothesis stated as follows:

H_0: Fault predictions obtained through ^{d}FPM are not significantly comparable/close to those obtained from industrial experiments.

H_a: Fault predictions obtained through ^{d}FPM are significantly comparable/close to those obtained from industrial experiments.

^{d}FPM's values of the project have been tested using the Chi-square test (χ^2). The χ^2-test applies only to discrete data, counted rather than measured values and hence becomes readily applicable in our context. The χ^2-test is not a measurer of the degree of relationship. It is merely used to estimate the likelihood that some factor other than chance (sampling error) accounts for the apparent relationship. Because the null hypothesis states that there is no relationship (the variables are independent), the test merely evaluates the probability that the observed relationship results from the chance. As in other tests of statistical significance, it is assumed that the sample observations have been randomly selected.

Table 3: Distribution of metric suite

	ENM	INH	CPM	COM
Maximum	3.00	1.0	3.00	3.00
Minimum	1.00	0.0	0.00	1.00
Median	3.00	1.0	0.00	2.00
Mean	2.45	0.5	1.00	1.54
Std Dev	0.30	0.5	0.60	0.79
Coefficients	-0.42 (C_0)	-1.4 (C_1)	0.24 (C_3)	1.90 (C_4)

Table 4: Model based fault prediction and comparison

Class	p-value	Model based prediction	Odd ratio	Experimental result	Comparative study
C_1	0.99	Faulty	99.00	Faulty	Same
C_2	0.87	Faulty	6.69	Faulty	Same
C_3	0.80	Faulty	4.00	Non-Faulty	Deviation
C_4	0.98	Faulty	49.00	Non-Faulty	Deviation
C_5	0.06	Non-Faulty	0.06	Non-Faulty	Same
C_6	0.76	Faulty	3.06	Non-Faulty	Deviation
C_7	0.20	Non-Faulty	0.25	Non-Faulty	Same
C_8	0.75	Faulty	3.00	Faulty	Same
C_9	0.86	Faulty	6.14	Faulty	Same
C_{10}	0.31	Non-Faulty	0.44	Non-Faulty	Same
C_{11}	0.31	Non-Faulty	0.44	Non-Faulty	Same

Table 5: χ^2 test observations

	High	Low	Total
^{d}FPM	7_A	4_B	11
Industry value	4_C	7_D	11
Total			22

Value of χ^2 is: 5.60

The Chi-square observations for the systems are listed in Table 5 by using Eq. 3 (Victor *et al*., 1996), applicable for small samples as frequencies of cells are fewer than 10. The assumptions made for ^{d}FPM values are low for less than or equal to four and high for greater than four and the degree of freedom may be calculated by using the formula df = (row-1)(column-1).

$$\chi^2 = \frac{N[|AD-BC|-N/2]^2}{(A+B)(C+D)(A+C)(B+D)} \quad (3)$$

In Eq. 3, A, B, C and D are being replaced by 7_A, 4_B, 4_C and 7_D, respectively. The computed value of χ^2 is greater than the critical value of χ^2 for 1 degree of freedom at 0.05 level of significance, which are 3.84. The test indicates that there is a significant relationship between the ^{d}FPM value and industry value for faults of the system at the 0.05 level of significance. Hence, the null hypothesis is rejected and it leads to the inference that ^{d}FPM gives the same result regarding faults for the system as it was obtained by industry people.

CONCLUSION

The proposed model ^{d}FPM may be used to predict faulty classes in early design phase. It is apparent from the empirical validation that this model can be used effectively for predicting faulty classes. Prior information regarding fault prone module leads to better planning and testing with less efforts and budget. Testing time and efforts may be reduced by using the proposed model. The limitation of the proposed model is that it is not a general model but specific for object oriented software. However it is very unlikely that there exist fault proneness model with general validity i.e., model that can accurately predict the faultiness of software module of every application (Denaro and Pezze, 2003).

REFERENCES

Abreu, F.B., 2001. Using OCL to formalize object oriented metrics definitions. INESC. Software Engineering Group ES007/2001. May.

Bansia, J. and C.G. Devis, 2002. A hierarchical model for object-oriented code in design quality assessment. IEEE Trans. Software Eng., 28 (1): 4-17.

Baroni, A.L., S. Braz and F.B. Abreu, 2002. Using OCL to Formalize Object-Oriented Design Metrics Definitions. QUAOOSE'2002. Malaga. Spain.

Basili, V. and D. Hutchen, 1989. An empirical study of a syntactic complexity family. IEEE Trans. Software Eng., 9 (6): 664-692.

Brind, L.C., P. Devanbu and W. Melo, 2004. An investigation in to coupling measures for C++. Proceedings of 19th International Conference on Software Reliability.

Collofello, J.S. and S.N. Woodfield, 1989. Evaluating the effectiveness of reliability assurance techniques. J. Syst. Software, 9 (3): 191-195.

Denaro and G.M. Pezze, 2002. An empirical evaluation of fault proneness model. ICSE Proceedings of 24th International Conference, pp: 241-251.

Denaro and G.M. Pezze, 2003. Towards Industrially relevant fault-proneness model. I. J. Software Eng. Knowledge Eng., 13 (4): 395-417.

Denaro, G., S. Morasca and M. Pizze, 2002. Driving models of software fault proneness. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, pp: 361-368.

Elrad, T., R. Filman and A. Bader, 2001. Article series on aspect oriented programming. Commun. ACM., 44 (10): 29-97.

Fenton, N. and M. Neil, 1999. A critique of software defect prediction models. IEEE Trans. Software Eng., 25 (5): 675-689.

Fenton, N.E. and S.L. Pfleeger, 1997. Software Metrics: A Rigorous and Practical Approach. 2nd Edn. International Thompson Publishing.

Gill, G. and C. Kemerer, 1991. Cyclomatic complexity density and software maintenance productivity. IEEE Trans. Software Eng., 17 (12): 1284-1288.

Kamia, T., S. Kusmoto and K. Inoue, 2005. Prediction of fault proneness at early phase in object oriented development. 2nd International Symposium OO Real-Time Distributed Computing, pp: 253-258.

Kanmani, S., V. Rhymend Uthariaraj, V. Sankaranarayanan and P. Thambidurai, 2006. Object-Oriented Software Fault Prediction Using Neural Networks. Elsevier.

Khan, R.A. and K. Mustafa, 2004. Quality estimation of object-oriented code in design phase. DQ., 4 (2): 14-17.

Li, H.F. and W.K. Cheung, 1987. An empirical study of software metrics. IEEE Trans. Software Eng., 13 (6): 697-708.

Munson, J. and T. Khoshgoftar, 1992. The detection of fault prone program. IEEE Trans. Software Eng., 18 (5): 423-433.

Reddy, C.S., K.V.S.V.N. Raju, V.V. Kumari and G.L. Devi, 2007. Fault-prone module prediction of a web application using artificial neural networks. Software Engineering and Applications (SEA 2007) Cambridge, MA, USA.

Rothermel, G., M.J. Harrold and J. Dedhia, 2000. Regression test selection for C++ programs. J. STVR., 10 (2): 77-109.

Rothermel, G., S. Elbaum, A. Malishevsky, P. Kallakuri and B. Davia, 2002. The impact of test suite granularity on the cost-effectiveness of regression testing. In: Proceedings of the 24th International Conference on Software Eng., pp: 230-240.

Rumbaugh, J. Michael Blaha and W. Lorenson, 2001. Object-Oriented Modeling and Design. PHI.

Silby, R. and V. Basility, 1991. Analyzing error prone system structure. IEEE Trans. Software Eng., 17 (2): 141-152.

Stojanovic, M. and K. El Emam, 2001. ES1: A Tool for Collecting Object-oriented Design Metrics. Institute for Information Technology. National Research Council Canada.

Victor, R. Basili, L.C. Briand and W.L. Milo, 1996. A validation of object-oriented design metrics as quality indicators. Technical Report. University of Maryland. Dep. of Computer Science. College Park. MD. 20742 USA.

Witten, I.H. and E. Frank, 2000. Data Mining: Practical Tools and Techniques with Java Implementation. Morgan Kaufman Publisher.