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

1R.A. Khan and 2K. Mustafa
1Department of Information Technology, Babasaheb Bhimrao Ambedkar University, Lucknow, UP, India
2Department 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 (OPM) 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 Hutchens, 1989; Gill and Kemerer, 1991; Li and Cheung, 1987; Silby and Basili, 1991; Denaro et al., 2002; Kamarani 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 Encapsulation, Inheritance, 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 of 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 inappropriate 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 identify 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 Eaman (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.

### 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 Pesze, 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 Pesze, 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 (Murson 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 Pesze, 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 Pesze, 2002).

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

\[
P = \frac{1}{1 + \exp \left( - (C_0 + C_iX_i + \ldots + C_nX_n) \right)}
\]

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.

\[ P = 4FPM = 1/1+e^{-\left(\text{ENM+INM+CPM+COM}\right)} \]  

(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 \textit{4FPM} requires the similar set of object oriented software project. The faulty and non-faulty classes were predicted using \textit{4FPM}. 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, \( \chi^2 \) test has been used for testing the null hypothesis stated as follows:

- \( H_0: \) Fault predictions obtained through \( 4FPM \) are not significantly comparable/close to those obtained from industrial experiments.
- \( H_1: \) Fault predictions obtained through \( 4FPM \) are significantly comparable/close to those obtained from industrial experiments.

\( 4FPM \)'s values of the project have been tested using the Chi-square test (\( \chi^2 \)). The \( \chi^2 \)-test applies only to discrete data, counted rather than measured values and hence becomes readily applicable in our context. The \( \chi^2 \)-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.

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 \( 4FPM \) values are low for less than or equal to four and higher than four and the degree of freedom may be calculated by using the formula df = (row-1)(column-1).

\[ \chi^2 = \frac{N \left( \text{AAD-BC-N21} \right)^2}{(A+B)(C+D)(A+C)(B+D)} \]  

(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 \( \chi^2 \) is greater than the critical value of \( \chi^2 \) 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 \( 4FPM \) 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 \( 4FPM \) gives the same result regarding faults for the system as it was obtained by industry people.
CONCLUSION

The proposed model ‘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