社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

UML メトリクスを利用したオブジェクト指向ソフトウェアの品質予測モデル

カマルゴ　クルズ　アナ　エリカ† 　落水浩一郎†

† 北陸先端科学技術大学院大学 情報科学研究科

Quality prediction model for object oriented software using UML metrics

CAMARGO CRUZ ANA ERIKA† and KOICHIRO OCHIMIZU†

† Japan Advanced Institute of Science and Technology, School of Information Science, Asihidai 1-1, Nomi, Ishikawa, 923-1292 Japan
E-mail: †{erika.camargo,ochimizu}@jaist.ac.jp

Abstract Several studies, in the field of object-oriented software quality, have been performed to define models for predicting fault-prone code. However, their predictions take place after the implementation phase, using design-complexity metrics measured from the code. The primary aim of this paper is to provide the foundations for building a model, that predicts fault-prone code in the early phases of the life cycle of the software, using UML metrics. We found that some UML metrics, approximations of traditional design-complexity metrics, can be acceptable predictors of fault-proneness, as they showed similar performance to the performance of those metrics measured from the implementation.

Key words software quality, fault-proneness prediction, object-oriented design metrics, UML metrics

1. Introduction

The use of efficient metrics and measurements enables us to create baselines to evaluate, to detect problems and to predict future trends within the elaboration process of certain products. Thus, we can improve quality of the product. In the field of software development, the measurement of software quality comes in, the best cases, right after the implementation phase.

The final objective of this research is to build a model that predicts quality of object oriented software, in terms of fault-proneness of code. Such predictions must take place in the early phases of the life cycle of the software.

Several empirical studies have been done in the field of fault-prone code prediction [3]〜[6]. Most of them have used design-complexity metrics for predicting fault-prone code. However, these metrics are measured after the classes have been implemented.

Since our goal is to predict fault-pronenenes in early phases of the life cycle of the software, we decided to use metrics taken from UML artifacts.

The first question raised is 'Which metrics from UML artifacts can be used to predict fault-prone code?', and the second question is 'Are these UML metrics good predictors of fault-prone code?'.

As a first attempt to predict fault-proneness, we thought of making use of UML artifacts to approximate those code metrics, which have been shown to be good predictors of fault-prone code in previous studies.

Moreover, we also need to determine which is the most suitable technique for predicting fault-proneness of code.

This paper is organized as follows (see Figure 1): Section 2 contains background on the field of quality software and metrics for object-oriented software. Section 3 compares and analyzes some studies related to the prediction of fault-prone code, which used code metrics to perform their predictions. Section 4 refers to the use of UML metrics for predicting fault-prone code. Section 5 draws the overall conclusion of this paper, and present plans for future work.

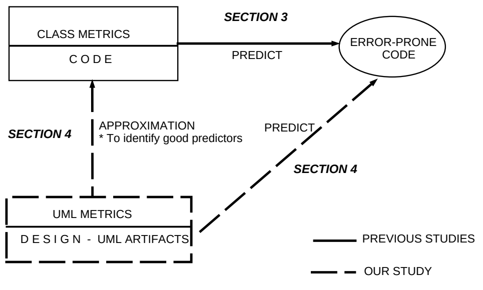


Fig. 1 Outline of this paper

2. Background

2. 1 Software Quality

In this paper we refer to software quality as the lack of bugs or defects in software. The more reliable software is, the higher its level of quality. Software reliability is one of the attributes considered in many quality software models [1].

In order to achieve reliable software, prevention of errors and detection and removal of faults in the software, must be practiced at early phases of its development. For this reason, prediction of fault-prone code before the implementation is desirable. Thus, we can save time and human resources.

2. 2 Design Complexity Metrics

The use of efficient metrics and measurements in any process gives us baselines to control, to predict future trends and to take decisions during the process. It is important not just to measure, but also to follow the proper guidelines of the metric used. In other words, we must be able to interpret the meaning of the metric correctly [2].

We can find several metrics for object-oriented software, some of the most used are: Weighted methods per class (WMC), Response for class (RFC), Lack of cohesion of methods (LCOM), Coupling between objects (CBO), Depth of inheritance tree (DIT), and Number of children (NOC). These metrics are known as the Chidamber and Kemerer (CK) metrics.

The CK metrics were defined to measure design complexity, and are traditionally measured from the implementation of the classes. Furthermore, they have been used in several studies to predict fault-proneness of code. In the next section, we introduce some of these studies.

3. Related work - Fault-prone code prediction using code metrics

This section aims to describe briefly some of the key works, we have found in the literature on fault-prone code prediction for object-oriented software. Table 1 summarizes the results of these studies.

Table 1 Studies on fault-prone code prediction: Results

Study	Single metrics as good predictors	Best models	Results
1	• RFC, DIT, NOC (+++) • CBO (++) • WMC (+)	DIT,RFC,NOC,CBO and Level of reuse of a class	Faulty classes detected 82% Correct classification 76%
2	• RFC and others coupling (+++) • NOC, DIT and others of inheritance (++)	4 coupling metrics plus 3 of inheritance	Faulty Classes detected 81% Correct classification 82%
3	No details	• 7 size metrics, 10 of cohesion and 29 of coupling • 7 size metrics, 29 of coupling and 18 of inheritance group	R-square=0.93 (A perfect fit gives: R-square=1)
4	• CK: RFC, WMC • QMOOD: NOM, CIS	• 3 CK Models • 1 QMOOD Model	• CK Models accuracy: 62.9 - 90.6 % • QMOOD Model accuracy: 63.5 - 82.5 % through the six versions of the SW analyzed

(+++) very significant (++) significant (+) somehow significant

(1) Basili et al.,1996. The goal of this study was to evaluate CK metrics as predictors of fault-prone classes. In order to validate their hypothesis, CK metrics were collected from 180 classes of various Management Information Systems developed in C++. Moreover, logistic regression was used to build a model that predicts fault-prone classes.

The most significant metrics for prediction of faulty classes were DIT, RFC and NOC, followed by CBO and WMC. Their prediction model was built using DIT, RFC, NOC, CBO and, as an additional metric, the level of reuse of the class. This model predicted correctly 76% of the overall classes, and it detected 48 faulty classes of 58, which is 82% of the faulty classes correctly predicted. They concluded that the CK metrics are useful predictors of fault-proneness [3].

(2) Briand et al.,2000. This work refers also to an empirical study for exploring the relationships between 28 metrics(among them CK metrics), and the probability of fault detection in classes during the testing phase. This study uses the same collected data as the study mentioned above in [3]. It also uses logistic regression for building prediction models.

Coupling measures that are counts of methods invocations as RFC, are found to be strongly related to the probability of fault detection in a class. Inheritance measures, such as NOC and DIT, were also found to be strongly related to fault-proneness detection. Multivariate analysis results show that by using some of the coupling and inheritance measures, very accurate models can be derived to predict fault-proneness of classes. Their best model obtained a percentage of correct classifications of about 82%, and 81% of correct detection of faulty classes, which draws a 94% of the total faults found in the entirely system [4].

(3) Kanmani et al.,2004. This study discusses the application of General Regression Neural Networks for predicting software quality in terms of fault-proneness of code through an empirical study. Object design measures of coupling, cohesion and inheritance, and size measures were collected from a Library Management System, and developed in C++; among these measures are the CK metrics. Fault-proneness of a class is measured as the fault ratio of the number of test cases not satisfied over the total number of test cases associated to the class.

They found two groups of metrics that perform well at predicting fault ratio of classes. The first group is the size-cohesion-coupling group, and the second group is size-coupling-inheritance group. Both groups obtained an R-square of about 0.93, which measures the accuracy of the model. A perfect fit of the model would result in an R-square value of 1 [5].

(4) Olague et al., 2007. This paper describes an empirical validation of three suites of metrics to predict fault-prone classes. These suites of metrics are: the CK suite, the Abreu's Metrics for Object-Oriented Design (MOOD) and Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). Data was collected from six versions of the Mozilla Rhino project, an open source software written in Java.

CK and QMOOD metrics were found to be good predictors of fault-proneness. The CK-RFC, CK-WMC and two metrics of the QMOOD set (Number of methods - NOM and Class Interface Size - CIS) were found to be consistent predictors of fault-prone code. Applying multivariate logistic regression, it was found that three CK models predict most accurately fault-proneness of code, followed by one QMOOD model. The accuracy of the CK models ranges from 62.6 to 90.6 percent through the six versions of Rhino. And for the the QMOOD model ranges from 63.5 to 85.2 percent [6].

To summarize, empirical studies have shown the ability of some object-oriented metrics to predict fault-proneness of code. The CK metrics are ones of most used, and ones of the best at predicting fault-prone code. Furthermore, Most of the results found coupling measures, such as the RFC metric, and inheritance measures, such as the DIT metric, strongly related to fault-proneness of code.

4. Fault-prone code prediction using UML metrics

We have already described some of the works done in the field of fault-proneness prediction for object-oriented software. These studies used design-complexity metrics measured from the code, thus later can perform their predictions. The final target of this research, is to predict fault-prone code before coding. Which UML metrics to use and how to make such predictions are the questions we attempt to answer in the following sections.

4. 1 Candidate UML metrics for predicting fault-proneness

At first we thought of approximating those design-complexity metrics, good predictors of fault-prone code, using UML artifacts. The main purpose of this approximation is to obtain good candidates for our prediction model.

As we mentioned in the previous section, several metrics have been used for predicting fault-prone code. In the results shown by Olague et al. [6], CK and QMOOD sets of metrics are found to be good predictors of fault-proneness. By studying the definition of these code metrics, we learned that most of them can be obtainable from UML class diagrams, except for the three CK metrics RFC, CBO and LCOM. The calculation of these metrics requires information within the bodies of the classes. For this reason, we make use of UML collaboration diagrams [1] as follows.

Response for Class (RFC)

Response for Classes has been defined as the set of all methods that can be invoked in response to a message to an object of the class. The response set of a class is the set of methods of the class plus all methods called by any method in the class. Thus, from the UML class diagrams we can learn the methods of the class, but not those methods called by any method in the class. For this purpose, UML collaboration diagrams can be useful. Below, we present two approaches to approximate RFC.

Approach 1: From the collaboration diagrams we sum all the different messages received, plus all the different messages sent, of every represented object of the class.

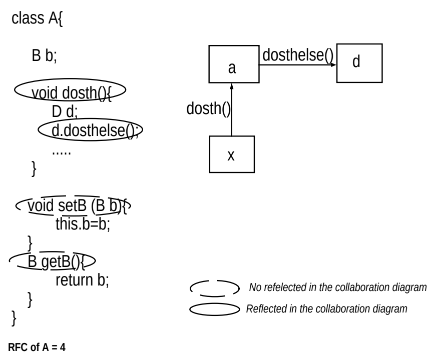
Approach 2: From the class diagrams we take the number of private attributes and we multiply this number by 2. If we sum this number to the number obtained by using the approach number 1, we will have included the setters and getters of the class, which are also public methods of the class, but which are not reflected in the UML collaboration diagrams, (See Figure 2).

Coupling between objects (CBO)

Coupling between objects is defined as a count of the number of other classes to which a class is coupled. If a method within a class uses a method or instance of a variable of a different class, It is said that this pair of classes is coupled.

Approach 1: From the UML collaboration diagrams, we

1} Collaboration diagrams for UML 1.5 and Communication diagrams for UML 2.0



```
class A{
   B b;
   void dosth(){
      D d;
      d.dosthelse();
      .....
   }
   void setB (B b){
      this.b=b;
   }
   B getB(){
      return b;
   }
}
```

RFC of A = 4
a
d
x
dosthelse()
dosth()
No refelected in the collaboration diagram
Reflected in the collaboration diagram

Fig. 2 RFC and Collaboration Diagrams

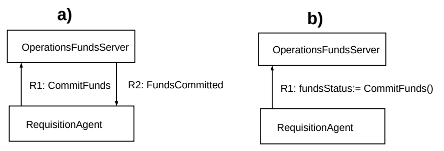


Fig. 3 Measuring CBO

count all the different messages sent by the objects of the class in the diagrams.

Approach 2: The same as approach Number 1 but eliminating those messages which return a value. Figure 3-a, represents a UML collaboration diagram where the object OperationsFundsServer is returning a value of FundsCommited to the RequisitionAgent object, represented by message R2. If we count R2 message as part of CBO for OperationsFundsServer, we would be assuming that OperationFundsServer is instantiating a method of RequisitionAgent, when it is just returning a value. Therefore, we propose using a different representation of this interchange of messages, as it is in Figure 3-b.

Lack of Cohesion Of Methods (LCOM)

LCOM measures the dissimilarity of methods in a class. This dissimilarity is measured via instance of variables or attributes within the methods. Unfortunately, we can not have this information from UML diagrams, unless details of every single method of all the classes, are specified.

Later in this paper, we will present an empirical study for the evaluation of these UML metrics, approximations of RFC and CBO metrics.

4. 2 Prediction Techniques

In this section, we discuss which is the most suitable technique for predicting faulty classes using UML metrics. There exist a number of statistical techniques that can be used for this purpose, and everything depends on the the kind of data we want to analyze.

Since we are looking for a technique to help us find the relationship between UML metrics acting as independent variables and fault-proneness of classes as the dependent variable, dependence statistical methods are the best candidates to apply. The two most used dependence methods for predicting fault-proneness of code are: Linear Regression and Logistic Regression. In the next paragraphs we discuss some characteristics of these techniques.

Linear Regression

Use: It is used when we have one or more independent variables (x) and one dependent variable (y). The dependent variable is a continuous metric. The independent variables cannot be linear combinations of each other. And the data used has a normal distribution.

Goal: To find an equation that best predict the value of the dependent variable (y), as a linear function of the independent variables (x_s), look at Equation 1, where y represents the expected value of the dependent variable for a given set of x_s values, x_s represents the independent variables, b_i is the estimated slope of a regression of y on x_i and a is the intercept.

$$y_{exp} = a + b_1x_1 + b_2x_2 + ...b_nx_n \quad (1)$$

Logistic Regression

Use: It is adequate when we have one or more independent variables and our dependent variable has just two values, in other words, when the dependent variable is dichtonomous (faulty/no faulty, 1/0).

Goal: The goal of Logistic Regression is to predict the probability of getting one of the two values of the dependent variable, given the independent variables. Logistic Regression uses the logit function to link the values of the independent variables to the probability of occurrence of one of the values of the dependent variable.

Looking at Equation 2, P is the probability of occurrence of one of the values of y, for example the probability of having a faulty class, x_s variables are the independent variables, b_i variable is the estimated slope of regression of P on x_i and a is the intercept.

$$P = \frac{1}{1 + e^{-(a+b_1x_1+b_2x_2+...b_nx_n)}} \quad (2)$$

Logistic Regression does not assume linearity of relationship between the independent and the dependent variables. And it does not require normally distributed variables.

Logistic Regression seems to be the more suitable technique for our problem, since no assumptions on the variables are made. Therefore, we have chosen to use Logistic Regression for the construction of our prediction model.

4. 3 Case Study Description

The goal of this preliminary experiment was to determine whether or not UML metrics are good metrics to predict fault-proneness of code. Since we are using UML metrics, approximations of code metrics, there is a possibility that those approximated metrics have lost or severely affected their prediction ability.

An e-commerce system was developed by 3 first-year-master students in the lecture "Software Design Laboratory" offered by The School of Information Science of The Japan Institute of Science and Technology. This e-commerce system is a small software system written in Java, and it has 13 main classes which were used for this study.

In order to test the prediction abilities of UML metrics referred to in Section 4.1, approximations of CBO and RFC metrics, we used univariate logistic regression to evaluate the relationships of each of these metrics and fault-prone classes. We also evaluated three other metrics of the QMOOD set, which have been found to be good at predicting fault-prone code [6] and can be easily obtained from UML class diagrams:

- Class Interface Size (CIS). It is a count of public methods in a class.
- Data Access Metric (DAM). The ratio of the number of private or protected attributes to the total number of attributes declared in a class.
- Number of Methods (NOM). The same as CK-WMC metric when all complexities are consider unity.

This experimental study basically consisted of:

(1) The collection of the RFC, CBO and the other 4 QMOOD metrics from UML diagrams of the e-commerce system.

(2) The collection of the same metrics from the implementation of the e-commerce system. For the CK metrics, the CKJM tool [2] was used.

(3) The collection of data referring to faults per class produced during the testing phase. We collected this information from the logs of the CVS repository used during the implementation of the e-commerce system mention above.

(4) Finally, the application of univariate logistic regression to predict the probability of having a faulty class using each of the UML and code metrics. Thus, we can determine if the proposed UML metrics are as good predictors of fault-prone classes as the code metrics are supposed to be. SAS software [3] was used for this purpose.

4.4 Case Study Results

4.4.1 UML metrics and code metrics linear relationship

Using the collected UML metrics and the code metrics from the e-commerce system mentioned above, we analyzed which of the UML metrics is a better approximation of the code metrics. This was done, by calculating the correlation coefficients between the code measures and the UML measures.

The correlation coefficients between the code RFC and the UML RFC Approximation 1 is -0.07, and between the code RFC and UML RFC Approximation 2 is 0.67.

The correlation coefficient between the code CBO and UML CBO Approximation 1 is 0.81, and between the code CBO measures and UML CBO Approximation 2 is of 0.89.

The results obtained indicate that the second approaches to approximate the code RFC and CBO metrics, have stronger linear relationships to the code metrics than the first approaches, so we expect these two approaches to perform closer to the performance of the code metrics at predicting fault-proneness of code.

4.4.2 UML metrics as good predictors of fault-prone code

In this Section, we will show the results obtained from the application of Univariate Logistic Regression technique for predicting faulty classes, using the UML metrics and the code metrics.

We present three indicators of correct classification:

- Correctness: The percentage of correct classifications of total of classes.
- Sensitivity: The percentage of correct classifications of just faulty classes.
- Specificity: The percentage of correct classifications of just no-fault classes.

Applying Univariate Logistic Regression over the data collected from the implemented classes of the e-commerce system and the faulty classes found in the testing phase, we obtained twelve different logistic models. Every model uses as independent variable each of the previously referred UML metrics. The univariate logistic equation is:

$$P = \frac{1}{1 + e^{-(A+Bx)}} \qquad (3)$$

Where P is the probability of having a faulty class, x is the independent variable, A is the coefficient slope and B is the regression coefficient of the independent variable x.

In Table 2 we show the different coefficients obtained for every different independent variable used.

Applying these models over the same data collected from the e-commerce system, which was also used for building the models, we obtained the results shown in Table 3. To summarize, we have that:

- The models using both of the UML approximations of CBO, obtained similar results. Their percentages of correct classifications are about 70%. While the model using the

2) This tool calculates CK metrics by processing the byte-code of compiled Java files. It was written by Diomidis Spinellis.

3) SAS stands for Statistical Analysis System.

Table 2 Univariate Logistic Regression Coefficients

Independent variable - x	A	B
CBO - code	-1.3863	8.3282
CBO - UML(1)	-0.4294	0.4955
CBO - UML(2)	1.22 e-12	6.0038
RFC - code	-12.32	0.4507
RFC - UML(1)	1.834	0.5808
RFC - UML(2)	-5.1106	0.4542
CIS - code	-20.8259	4.0259
CIS - UML	-8.6531	2.0077
DAM - code	-0.1476	0.9682
DAM - UML	-12.3695	19.0238
NOM - code	-24.8429	3.9561
NOM - UML	-3.3616	0.6889

Table 3 Results of univariate logistic models

Independent variable x	Correctness	Sensitivity	Specificity
CBO-code	92.3%	88.88%	100%
CBO-UML(1)	69.2%	66.66%	75%
CBO-UML(2)	69.2%	55.55%	100%
RFC-code	84.61%	88.88%	75%
RFC-UML(1)	76.92%	77.77%	75%
RFC-UML(2)	84.61%	88.88%	75%
CIS-code	90.9%	85.7%	100%
CIS-UML	90.9%	100%	75%
DAM-code	36.3%	57.14%	0%
DAM-UML	72.7%	85.7%	50%
NOM-code	90.9%	85.7%	100%
NOM-UML	72.7%	71.42%	75%

CBO measures from the code obtained percentages of about 90%.

- In the case of the models using RFC measures, the model using approach number 2 to approximate RFC performed equally to the model using the RFC measures from the code.

- The model using UML CIS measures performed similarly to the model using the code measures.

- The model using code DAM measures indicates that DAM is not related to fault-proneness.

- The model using UML NOM measures obtained percentages of correct classification of about 70%, while the model using code measures obtained percentages of correct classification of about 90%.

5. Conclusion and Future work

As a first attempt to predict fault-prone code before its implementation, we analyzed how CK and QMOOD metrics, good predictors of fault-prone code, can be obtained from UML diagrams. Most of these metrics can be measured from UML class diagrams, except for RFC, CBO and LCOM. By using UML collaboration diagrams, we approximated the values of RFC and CBO metrics. Moreover, we analyzed the strength of five of these UML metrics at predicting faulty classes. We found that one of our approaches to approximate RFC with UML diagrams performs equally to the RFC metric measured from the code in models that relates RFC to the probability of having a faulty class. Also, we found that the model using CIS measures from UML class diagrams, performs similarly to the model using code measures. The performance of the other UML approximations somewhat were acceptable (including the UML metrics for CBO). We concluded that UML metrics can be acceptable predictors of fault-proneness.

Further study on UML metrics is needed, as well as, an evaluation of their effectiveness as predictors of fault-prone code. Special focus should be on those metrics that involve inheritance and coupling measures, since they have been found to be strongly related to fault-proneness. There is some ongoing-work on formal definition of UML metrics, which we would like to study deeply [7].

Furthermore, Multivariate logistic regression must be applied, in order to find a robust model for predicting fault-prone classes. Such experiment requires lots of information, which we expect to count with, in a near future.

Bibliography

[1] Rosenberg Linda, Hammer Ted, and Shaw Jack, *Software Metrics and Reliability*, IEEE International Symposium on Software Reliability Engineering, Germany, 1998.

[2] Rosenberg Linda, *Applying and Interpreting object oriented metrics*, Software Technology Conference, Utah, April 1998.

[3] Victor R. Basili and Lionel C. Briand and Walcélio L. Melo, *A Validation of Object-Oriented Design Metrics as Quality Indicators*, IEEE Transactions on Software Engineering, Piscataway, NJ, USA, October 1996.

[4] Lionel C. Briand, Jürgen Wüst, John W. Daly and D. Victor Porter, *Exploring the relationships between design measures and software quality in object-oriented systems* Journal of Systems and Software,2000.

[5] Kanmani, S., and Uthariaraj V. Rymend, *Object oriented software quality prediction using general regression neural networks*, SIGSOFT Soft. Eng. Notes, New York NY, USA, 2004.

[6] Hector M. Olague and Sampson Gholston and Stephen Quattlebaum, *Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes*, IEEE Transactions on Software Engineering, Piscataway, NJ, USA, 2007.

[7] Aline Lúcia Baroni, *Quantitative Assessment of UML Dynamic Models*, SIGSOFT Soft. Eng. Notes, ACM, New York NY, USA, 2005.