On the Prediction of Fault-proneness in Object-Oriented Development

Toshihiro Kamiya, Shinji Kusumoto and Katsuro Inoue

Graduate School of Engineering Science, Osaka University, Japan
1-3 Machikaneyama, Toyonaka, Osaka
560-8531, Japan
Tel: +81-6-6850-6571
Fax: +81-6-6850-6574
E-mail: kamiya@ics.es.osaka-u.ac.jp

ABSTRACT
This paper proposes a new method to estimate the fault-proneness of the class in the early phase, using several complexity metrics for object-oriented software.

Keywords
Metrics, object modeling technique, object-oriented development, fault estimation

1 INTRODUCTION

In attempt to reduce the number of delivered faults, it was reported that most companies spend between 50-80% of their software development effort on testing[4]. Therefore, reducing the effort of testing is a key to high productivity in software development. Software review is one of the most effective techniques to reduce the testing effort. Reviews can detect many faults before testing, with lower cost compared to testing.

In order to effectively review and test the software products, it is needed to identify the fault-prone modules so that review and testing effort can be concentrated on the modules[1]. Chidamber and Kemerer proposed six complexity metrics for object-oriented software to evaluate the main characteristics which only object-oriented software essentially possess [3].

However, in [1]and [3], the metrics were applied to the source code. The estimation of the fault-proneness should be done in the early phase to effectively allocate the effort for fixing the faults.

This paper proposes a new method to estimate the fault-proneness of the class in the early design phase, using several complexity metrics for object-oriented software. In the proposed method, we introduce four checkpoints into the analysis/ design/ implementation phase based on OMT[9].

2 PROPOSED METHOD

We regard the analysis/ design/ implementation phase as a series process in which the information about software product gradually increase as the process progresses.

We introduce four checkpoints in the development process from the viewpoint of measurement and identify which information has been added to the design specification at each checkpoint. We define the subset of the conventional metrics applicable to the design specification developed at each checkpoint. Then we estimate the fault-proneness of the class using the multivariate logistic regression analysis with the applicable metrics. We employed Chidamber and Kemerer's six metrics[3] and NIV[7] as object-oriented design metrics(see Table 2).

(CP1)Entity and relation:

At the CP1, reference-relationship between classes and attributes of the classes have been determined. Reference-relationship corresponds to the coupling between classes and the attribute corresponds to the instance variable. NIV can be calculated from the attribute information. Though CBO can be calculated from the reference-relationship between classes, reference to the reused class in the class library is not clearly described and thus the value of CBO is not correct.

(CP2)Structure and inheritance

At the CP2, derivation-relationship between classes and the methods in the classes have been determined. In order to determine derivation-relationship, the class hierarchy tree is clearly described. Thus, DIT can be calculated from the derivation-relationship. WMC can be calculated from the information of the methods. Since the reused classes are determined, the value of CBO can be calculated correctly.

(CP3)Algorithm

At the CP3, algorithms in each method and call-relationship between the methods are determined. Based on the information, LCOM and RFC can be calculated.

(CP4)Implementation

CP4 is the checkpoint where source code has been implemented. For each class, SLOC (Source Lines Of Code) can be calculated.

Table 1: Checkpoint and available metrics

Checkpoint	Added Information	Available Metrics
(CP1)Entity and relation	Reference-relationship among classes, attributes of class	NIV, CBON
(CP2)Structure and inheritance	Class hierarchy, methods, reused library	NIV, CBON, CBOR, CBO, WMC, DIT, NOC
(CP3)Algorithm	Algorithm of the method	NIV, CBON, CBOR, CBO, WMC, DIT, NOC, RFC, LCOM
(CP4)Implementation	Source code	NIV, CBON, CBOR, CBO, WMC, DIT, NOC, RFC, LCOM, SLOC

CBO, DIT, LCOM, NOC, RFC, and WMC are Chidamber and Kemere's metrics[3]. NIV is number of instance variables in the classes[7]. CBON is CBO for Newly-developed classes, and CBOR is CBO for Reused classes (see text of the paper).

CBO can be calculated in the CP1 and the CP2. CBO is to count the number of coupling between the target class and other classes. Since CBO at CP1 only counts the number of coupling between the target class and the classes developed from scratch, the value of CBO is not calculated in the way of original definition. However, our previous research result showed that CBO at CP1 have highly correlated with the fault-proneness[6]. So, we introduce the following two metrics.

CBON(Coupling Between Object classes Newly-developed)

CBON is the number of coupling between the target class and the classes developed from scratch.

CBOR(Coupling Between Object class Reused)

CBOR is the number of coupling between the target class and the reused classes

From the definitions, CBO is the sum of CBON and CBOR.

3 EMPIRICAL EVALUATION

Overview

The experimental project was performed at a computer company for five days in August 1997. The main characteristics of the project can be summarized as follows:

(1) The Developers were new employees of the computer company and had just graduated from college in March 1997.

(2) There are sixteen developer teams and each team built a mail deliver system of an identical requirement. Each team consisted of four or five developers.

(3) The programs were implemented in C++ language.

(4) Each developer worked on the assigned PC (Personal Computer). All PCs were connected to the PC server via intranet. The server collected the source code files of the developers every one hour.

We collected complexity metrics and fault data from each developer. Unfortunately, we could not collect the design specification in this experiment. So, according th eassumption that all informations of the design specification are included in the source code, we collected the metrics values from source code by using a metrics tool.

Analysis

Table 2: Fault Predict Precision at Checkpoint

Checkpoint	CP1	CP2	CP3	CP4
Correctness(%)	82	76	85	86
Completeness(%)	33	59	63	70
Error-based Completeness(%)	46	75	71	83

Table 2 shows the estimation accuracy of the fault proneness by mutivariate logistic regression analysis at each checkpoint of (CP1)..(CP4). The correctness is the percentage of classes correctly predicted as faulty(the number of predicted faulty and actually faulty classes / the number of predicted faulty classes). The completeness is the percentage of faulty classes detected(the number of predicted fault and actually faulty classes / the number of actually faulty classes). The error-based completeness is the percentage of faults found in classes correctly predicted as faulty.

Completeness at CP1 is relatively low (33%). On the other hand, correctness is high(82%). Thus, the estimation can be used to "seed" the classes in which faults would be introduced. The seeded classes become the

candidates that should be reviewed and tested selectively. Also, the location of the seeded classes might be the criterion of the judgment for review. For example, if the seeded classes are concentrated on the important section of the design specification and the section is difficult to test, we should redesign it.

For completeness, estimation at CP2 is excellent with respect to the upper limit at CP4. Though the metrics for algorithms of the method cannot be used at CP2, the result is a surprising one. It suggests that it would be possible to estimate the fault-proneness from the design specification in the design phase where the algorithms are not determined, without source code.

The result of estimation at CP3 fell short of our expectations. Compared to the estimation result at CP2, the accuracy is not improved very much. We consider that the accuracy would be improved by using 'fine-grained' C++ design metrics[2] together at CP3. Chidamber also mentioned that the calculation of WMC depends on the implementation of the target method. In this experiment, according to the [1] and [3], we assumed that the complexity of the method is unity. It would improve the accuracy at CP3 that the WMC weighted with traditional metrics (such as Cyclomatic Number[8] or Software Science[5]).

4 CONCLUSION

In this paper, we proposed a new method to estimate the fault-proneness of the class in the early design phase, using several complexity metrics for object-oriented software. In the method, we have introduced four checkpoints into the analysis/ design/ implementation phase, in which particular subsets of metrics are applicable. We have also applied the proposed method to an experimental project. The analysis result shows the validity and usefulness of the proposed method.

ACKNOWLEDGEMENTS

We would like to thank the support of Mr. Yukio Mohri and Masahiro Takahashi of Nihon Unisys corporation for the experimental projects in Section 3. We also thank Mr. Shuji Takabayashi of Nara Institute of Science and Technology for his assistance in building a metrics tool.

REFERENCES

[1] V. R. Basili, L. C. Briand, and W. L. Melo: "A validation of object-oriented design metrics as quality indicators", IEEE Trans. on Software Eng. Vol. 20, No. 22, pp. 751-761 (1996).

[2] L. C. Briand, P. Devanbu, and W. Melo: "An Investigation into Coupling Measures for C++", Proc. of the19th Int'l Conference on Software Eng., Boston, USA, pp.412-421(1997).

[3] S. R. Chidamber and C. F. Kemerer:"A metrics suite for object-oriented design," *IEEE Transactions on Software Engineering*, Vol.20, No.6, pp.476-493(1994).

[4] J. S. Collofello and S. N. Woodfield: "Evaluating the effectiveness of reliability-assurance techniques," *Journal of Systems & Software,* Vol.9, No.3, pp.191-195 (1989).

[5] M. H. Halstead: *Element of software science*, New York, Elsevier North-Holland(1977).

[6] T. Kamiya, S. Kusumoto, K. Inoue, and Y Mohri: "Empirical Evaluation of Reuse Sensitiveness of Complexity Metrics", *Information and Software Technology*, (Accepted).

[7] M. Lorenz and J. Kidd: *Object-Oriented software metrics*, New Jersey, Prentice Hall(1994).

[8] T. J. McCabe: "A complexity measure," *IEEE transactions on software engineering*, Vol.SE-2, No.4, pp.308-320(1976).

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen: *Object Oriented Modeling and Design*, Prentice Hall(1991).