A Straightforward Approach to Effort Estimation for Updating Programs in Object-Oriented Prototyping Development

Satoru Uehara, Osamu Mizuno and Tohru Kikuno
Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University, Japan
{s-uehara, o-mizuno, kikuno}@ics.es.osaka-u.ac.jp

Abstract

In this paper we discuss estimation of efforts needed to update programs according to a given requirement change. In the Object-Oriented prototyping development(shortly the OO prototyping), the requirement changes occur frequently and regularly. Thus a simple and fast estimation of efforts is strongly required by both developers and managers. However, existing estimation methods cannot be applied to the OO prototyping.

Therefore we will try to propose a straightforward approach to effort estimation, which reflects the specific properties of the OO prototyping. First, we analyze the following characteristics of the OO prototyping: (1) updating activities consist of creation, deletion and modification, (2) the target to be updated has four kinds of types(void type, basic type, library type and custom type) and (3) the degree of information hiding is classified into private, protected and public.

Then, we present a new formula $E(P, \sigma)$ to calculate the efforts needed to update a program P according to a set of requirement changes σ. The formula $E(P, \sigma)$ includes weighting parameters: w_{upd}, w_{type} and $w_{inf\text{-}h}$ according to the characteristics (1), (2) and (3), respectively. Finally, we conduct experimental evaluations by applying the formula $E(P, \sigma)$ to actual project data in a certain company. The evaluation results prove statistically to some extent the validity of the proposed approach.

1 Introduction

For the software development, a lot of development paradigm has been proposed. Among them, the Object-Oriented(OO) development, shortly the OO development, has such high capability that transforms the real complex things into software products using the concept of object. The environments of development of software products in the OO language such as Smalltalk, C++ and JAVA have been provided, and various techniques for the OO development have been proposed[3, 12]. As a result, the OO development has been widely used in industries.

It is also said that the OO development is reasonable and natural to be combined with rapid prototyping[10, 13]. Because the OO paradigm makes it easy to understand the system structure and to reuse the previous components of other systems. We call the rapid prototyping combined with the OO development as the Object-Oriented prototyping development(shortly, the OO prototyping).

In the OO prototyping, it is very important from the management point of view to estimate the cost for updating needed by each requirement change. Since the requirement changes are issued frequently from the customers, the developers have to update or revise so often their design or program code to prepare a new version for new requirement.

Based on an accurate estimation of successive updating activity, the developers can achieve not only high productivity but also high quality of the program code[11]. So managers have to know how much their updating activities are needed. In this paper we try to estimate the cost for updating activities using the only data that can be obtained before coding activities start.

Although there are many models and methods for cost estimation in software development, such as COCOMO model[2] and the function point method[1], there are little methods which take the property of the OO development into account. Furthermore, since the existing methods need a lot of preparation with respect to customization of methods to the environments, it is difficult to apply in such a small development that must be done under the restrictive cost limitations.

Our objective is to propose a straightforward approach to estimate efforts for updating programs in the OO prototyping development. In other words, we aim to develop an intuitive estimation method which is easily and cheaply applicable to the real development environment. In the method, we

present a formula $\mathbf{E}(P, \sigma)$ for the effort estimation where P is a certain version of program to be updated and σ is a set of requirement changes. The $\mathbf{E}(P, \sigma)$ is calculated as the sum of scores for all activities included in updating.

In order to define the efforts $\mathbf{E}(P, \sigma)$, we have analyzed the activities for updating a program. First, we clarify the following three viewpoints: (1) the kind of updating activities, (2) the types of targets to be updated, and (3) the degree of information hiding. Then we define the formula $\mathbf{E}(P, \sigma)$ using three weighting parameters: w_{upd}, w_{type} and $w_{inf\text{-}h}$ introduced to consider (1), (2) and (3), respectively.

Finally we perform experimental evaluation of our proposed method. We apply the formula $\mathbf{E}(P, \sigma)$ to two actual project data in a certain company. The statistical analysis on the results of these experiments shows to a certain extent the validity of our proposed method.

The rest of this paper is organized as follows: Section 2 shows an outline of our work. The key idea of our study is explained in Section 3. The proposed cost estimating method is shown in Section 4, and sample values for weighting parameters are given in Section 5. Section 6 shows two experiments for evaluation. Finally Section 7 concludes this paper.

2 Effort Estimation for Prototyping

2.1 Prototyping Development

Figure 1 shows an outline of typical prototyping process[10]. Generally speaking, the prototyping development process consists of the rapid interactions between customers and developers. One of specific characteristics of the prototyping is that any complete or fixed specifications do not exist during the development. The developers design and implement the product based on the customers' requirements, and deliver the prototype of program to the customers. Then the customers try to test the prototype, and return the changes of requirements to the developers. The development is iterated until the customers satisfy the program.

Generally speaking, it is very reasonable and natural to combine the prototyping development and the object-oriented development. Since the object-oriented paradigm makes it easy to understand the system structure and reuse the previous components of other systems, the developers can deliver the product rapidly using the advantages of object-oriented development. This kind of speed-up provides a good advantage to prototyping development.

In the afterwards, we call the prototyping development combined with the Object-Oriented development as the Object-Oriented prototyping development(shortly, the OO prototyping).

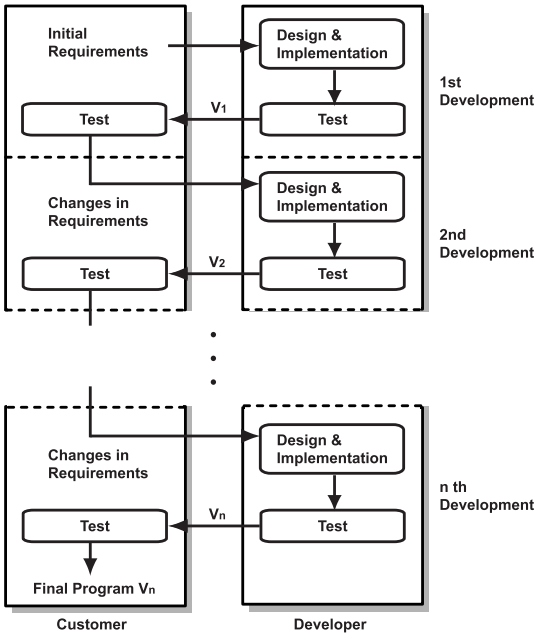

Figure 1. Development process

2.2 Our Objective

Our objective is to estimate the efforts for updating programs in the OO prototyping development. Since the requirement changes occur frequently and regularly, the cost estimation for updating activities should also be done very fast and be easy to apply. Although the accurate cost estimation helps increasing quality and productivity, the developers in the industry really want to use such an estimating method that is intuitively constructed and is easy to apply.

We aim to develop an estimation method considering the following properties from the practical point of view: (1) intuitive calculation: the way of estimation is intuitive for developers or managers, (2) quick calculation: the developer and manager can obtain the result of estimation immediately, and (3) easy calculation: the estimation method is easy to apply in the actual development environment.

2.3 Related Works

The traditional model such as COCOMO[2] is one of good solutions for cost estimation in general software development. However, our objective mentioned in subsection 2.2 is slightly different from that of the COCOMO. Since the COCOMO is a regression curve based model and aims to accurate estimate, it is difficult for general developers to understand and apply it.

On the other hand, Kusumoto et al. developed a simulator to estimate the efforts and faults of the software project[7].

However, their simulator was targeted to the standard waterfall model, so it is hard to apply the simulator to the prototyping development.

Our proposed method(to be defined in Section 4) is based on the score given to each fundamental component according to analysis of some characteristics of the OO programs. So it is very similar to the function point method[1] in that sense. However, the function point method is not easy to apply the OO development directly, since the function point method must deal with the number of inputs/outputs or the number of screens.

Therefore we have to establish a new straightforward(that is, intuitive, quick and easy) method to estimate the efforts for the object-oriented prototyping development.

3 Three Essential Characteristics

3.1 Updating Activities

In the OO prototyping development, the target to be updated could be classes, attributes in classes, or methods in classes. On the other hand, the activities for updating them are classified into three fundamental operations:

1) Creation: new creation of the target(that is, a class, an attribute in a class and method in a class).

2) Deletion: deletion of the existing target(that is, the existing class, the existing attribute or the existing method).

3) Modification: modification to the existing target.

Any modification can be expressed by one of three operations or the combination of them. For example, consider a case that an attribute of a class is newly created. Then the operation is expressed as creation. Next, consider a case that a parameter of a method is added. Then a sequence of the operations can be considered: an existing method is deleted and a new method with a new parameter is created. Thus it is expressed as combination of creation and deletion.

In the formulas for efforts estimation in Section 4, we distinguish them, and assign distinct score to each operation.

3.2 Types of Target to be Updated

Generally, the target(that is, the attribute or the method in a class in this subsection) to be updated has a type, and the effort to update it varies with its type. The types are categorized as follows:

1) Void type: Void type is a type which has no return value. In C++ and JAVA, `void` is an example.

2) Basic type: Basic type is a type which is originally installed in the OO language. In C++ and JAVA, `int` and `char` are examples.

3) Library type: Library type is a type which is defined in the class library. The classes in Microsoft Foundation Classes(MFC) are examples.

4) Custom type: Custom type is a type which is defined in the developing program.

In the formulas for efforts estimation, we distinguish these types and assign distinct score to each type.

3.3 Degree of Information Hiding

The Object-Oriented paradigm has several useful properties for a large scale software development. Among them, encapsulation is one of the most important properties in the OO paradigm. The degree of information hiding clearly affects the updating efforts, since the scope of variables varies according to it. The following summarizes information hiding in C++ and JAVA:

1) private: the attribute/method in a class cannot be referred by any other classes.

2) protected: the attribute/method in a class cannot be referred by any other classes except for its child class.

3) public: the attribute/method in a class can be referred and used by other classes.

4 New Formula for Effort Estimation

4.1 A New Formula

Based on the classification in Section 3, we introduce new formula to calculate the effort needed for updating of the OO programs.

Assume that P is a program to be updated according to requirement changes σ. In more precise, σ is a set of n requirement changes $R_1, \cdots, R_n$. Then P' is a resultant program that is obtained from P by executing necessary operations specified in $R_i (1 \leq i \leq n)$. In this paper, we propose the following formula to estimate the efforts for updating program P:

$$\begin{aligned}
\mathbf{E}(P, \sigma) &= \sum_{i=1}^{n} E_{requirement}(R_i) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} E_{class}(C_j) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} (\sum_{k=1}^{p} E_{attribute}(A_k) + \sum_{l=1}^{q} E_{method}(M_l))
\end{aligned}$$

Intuitively speaking, according to requirement change R_i, a class C_j is updated. In more detail, an attribute A_k in the class C_j and M_l in the class C_j are updated. We will explain the definition of E's in the successive sections.

4.2 Efforts for Updating

1) Efforts to meet a requirement change R

Assume that m classes $C_1, \cdots, C_m$ are updated for the requirement change R. The formula to estimate the efforts for requirement change R is defined as follows:

$$\mathbf{E}_{requirement}(R) = \sum_{j=1}^{m} E_{class}(C_j) \quad (1)$$

2) Efforts to update a class C

Assume that updating activity of a class C consists of updating p attributes $A_1, \cdots, A_p$ and q methods $M_1, \cdots, M_q$. The formula to estimate the efforts for updating the class C is defined as follows:

$$\mathbf{E}_{class}(C) = \sum_{k=1}^{p} E_{attribute}(A_k) + \sum_{l=1}^{q} E_{method}(M_l) \quad (2)$$

3) Effort to update a method M

Assume that a method M is updated. The formula to estimate the efforts for this update is defined as follows:

$$\mathbf{E}_{method}(M) = \alpha \times w_{upd} \times w_{type} \times w_{inf\text{-}h} \times (1 + WCC(M) + WCM(M) + WPM(M)) \quad (3)$$

where variables α, w_{upd}, w_{type}, $w_{inf\text{-}h}$ are constants to represent characteristics in Section 3 and WCC, WCM, WPM are fundamental OO metrics to be defined in the next subsection. The semantics of these variables are summarized as follows:

a) α: a basic score for updating a method.

b) w_{upd}: a weight representing the difference in the difficulty caused by the kinds of updating activities(see subsection 3.1).

c) w_{type}: a weight representing the difference in the difficulty caused by the type of return value(see subsection 3.2).

d) $w_{inf\text{-}h}$: a weight representing the difference in the difficulty caused by the degree of information hiding(see subsection 3.3).

4) Efforts to update an attribute A

Assume that an attribute A is updated. The formula to estimate the efforts for this updating is defined as follows:

$$\mathbf{E}_{attribute}(A) = \beta \times w_{upd} \times w_{type} \times w_{inf\text{-}h} \quad (4)$$

The semantics of variables are summarized as follows:

a) β: a basic score for updating an attribute.

b) w_{upd}: a weight representing the difference in the difficulty caused by the kind of updating activities(see subsection 3.1).

c) w_{type}: a weight representing the difference in the difficulty caused by the type of attribute(see subsection 3.2).

d) $w_{inf\text{-}h}$: a weight representing the difference in the difficulty caused by the degree of information hiding(see subsection 3.3).

When we apply proposed formula to actual development, we have to determine the values of α, w_{upd}, w_{type}, $w_{inf\text{-}h}$ in formula (3) and β, w_{upd}, w_{type}, $w_{inf\text{-}h}$ in formula (4). The sample values of these variables will be shown in subsection 5.2.

4.3 Fundamental Metrics

Until now, although various OO metrics have been suggested[4, 5, 6, 8, 9], most of them are for a class, not for a method. So, we suggest simple new metrics WCC, WCM, WPM which indicate complexity of a method.

1) **WCC**(Weighted Coupling Classes)

Assume that a method M includes n classes $C_1, C_2, \cdots, C_n$ which are referred in M. We introduce the weight wC_i for a class C_i, and then define **WCC** as follows:

$$\mathbf{WCC}(M) = \sum_{i=1}^{n} wC_i$$

In fact, the formula of **WCC** is a part of the definition of existing OO metrics *CBO*[5].

2) **WCM**(Weighted Coupling Members)

Assume that a method M refers n attributes $A_1, A_2, \cdots, A_n$ which are defined in a class C. We introduce the weight wA_i for an attribute A_i, and then define **WCM** as follows:

$$\mathbf{WCM}(M) = \sum_{i=1}^{n} wA_i$$

3) **WPM**(Weighted Parameters of Method)

Assume that a method M includes n parameters $P_1, P_2, \cdots, P_n$. We introduce the weight wP_i for a parameter P_i, and then define **WPM** as follows:

$$\mathbf{WPM}(M) = \sum_{i=1}^{n} wP_i$$

5 Case Study

5.1 Parameters to be Calculated

When a requirement change occurs, the developers in the first place decide which parts of the existing program are to be updated. The decision is performed based on the analysis results of various Object-Oriented design documents. Although they never know how many lines of codes are to be updated, they can get the following data (1)-(3) from the analysis. The data give the basis for the evaluation of $E(P, \sigma)$.

(1) Classes to be updated (that is, classes $C_1, \cdots, C_m$ in formula (1))

(2) Attributes to be updated in each class C_i(that is, attributes $A_1, \cdots, A_p$ in formula (2))

- 2-1) The kind of updating activity
- 2-2) The type of the attribute
- 2-3) The degree of information hiding

(3) Methods to be updated in each class C_i(that is, methods $M_1, \cdots, M_q$ in formula (2))

- 3-1) The kind of updating activity
- 3-2) The type of the method
- 3-3) The degree of information hiding
- 3-4) Concerning the metrics WCC, WCM, WPM, the followings are defined with respect to the external reference:
  - 3-4-1) The number of parameters and their types
  - 3-4-2) The number of references to external classes and their types
  - 3-4-3) The number of references to attributes in the class C_i and their types

By collecting above information, we can apply the proposed effort estimation to the project.

5.2 Sample Values for Weight

Here, we determine the values of weights explained in subsection 4.2. The values are shown in Table 1. These values of weights are obtained by means of questionnaires to the developers in a certain company. Since these weights are not determined by formal way, we have to evaluate the validity of values by some experiments in Section 6.

Table 1(a) shows values of w_{upd}, w_{type}, $w_{inf\text{-}h}$ for attribute A_i. In this case we set basic score $\alpha = 1.0$. Next Table 1(b) shows values of w_{upd}, w_{type}, $w_{inf\text{-}h}$ for method M_i. In this case we set basic score $\beta = 1.0$. Then Table 1(c) shows values of wC, wA, wP for software metrics.

Table 1. Sample weight

w_{upd}	Creation	Deletion	Modifi.
	1.0	0.2	N/A
w_{type}	Basic	Library	Custom
	1.0	1.1	1.2
$w_{inf\text{-}h}$	Private	Protected	Public
	1.0	1.0	1.0

(a) Attribute (Basic score α = 1.0)

w_{upd}	Creation	Deletion	Modifi.	
	1.0	0.2	0.8	
w_{type}	Void	Basic	Library	Custom
	1.0	1.0	1.2	1.5
$w_{inf\text{-}h}$	Private	Protected	Public	
	0.8	0.8	1.0	

(b) Method (Basic score β = 2.0)

	Basic	Library	Custom
wC	0.01	0.01	0.01
wA	0.01	0.01	0.01
wP	0.02	0.03	0.03

(c) Metrics WCC, WCM, WPM

6 Experimental Evaluation

In order to evaluate the validity of proposed method from two distinct aspects, we have two experiments based on different data sets. In Experiment 1, we use a set of JAVA programs. We evaluate the correlation between the effort obtained by proposed method and actual one for each updating activity. Next, in Experiment 2, we use a set of C++ programs. Then we evaluate whether the effort obtained by proposed method for each class reflects the difficulty of class, or not.

6.1 Experiment 1

6.1.1 Collected JAVA Programs

In the first experiment, the target project is a development of application that shows graphical images stored in Object-Oriented database. The program is written in JAVA, and it can run on WWW browser. The development was performed according to the OMT[12], and 10 versions of programs were successively developed.

Table 2 shows data which are collected from the project. In the table $V_i (i = 1, 2, \cdots, 10)$ denotes the *i*th version, # of classes denotes the number of classes included in V_i. The value of LOC implies the lines of code excluding the comments, and the value of *Person-Days* indicates the efforts needed to develop each version.

Table 2. Data from JAVA project

Version	# of classes	LOC	Person-Days
V_1	157	9197	445
V_2	169	25603	98
V_3	170	27845	90
V_4	187	28677	40
V_5	190	28916	76
V_6	188	28324	56
V_7	188	28524	54
V_8	189	28713	42
V_9	188	28722	42
V_{10}	189	28737	60

6.1.2 Effort Estimation using $\mathbf{E}(P, \sigma)$

In this experiment, we will investigate the correlation between efforts obtained by proposed method and actual efforts. Thus we calculate the updating effort $\mathbf{E}(P, \sigma)$ for all versions $V_1, \cdots, V_{10}$. The result is summarized in Table 3.

6.1.3 Statistical Analysis

Here we perform the correlation analysis between estimated efforts $\mathbf{E}(P, \sigma)$ in Table 3 and actual *Person-Days* in Table 2. Note that we exclude the data of V_1 from the correlation analysis. Since our objective is to estimate the efforts for updating activity, we have to exclude V_1 because it is a new development of programs in the project.

The correlation coefficient is calculated as 0.68. Figure 2 shows the relationship between them.

From the result of analysis, we can say that there is high correlation between efforts calculated by proposed formula and actual ones.

Table 3. Efforts estimation for version

Version	E(P,σ)
V_1	3523.7
V_2	1084.7
V_3	632.9
V_4	528.1
V_5	375.5
V_6	715.5
V_7	209.3
V_8	90.0
V_9	31.6
V_{10}	22.1

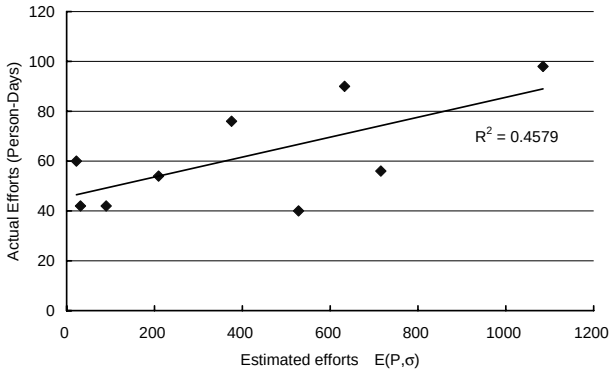

Figure 2. Actual efforts and $E(P, \sigma)$

6.2 Experiment 2

6.2.1 Collected C++ Programs

In the second experiment, the target project is a development of application for a banking related system[14]. The program was written in C++ on Microsoft Windows 95/98/NT. The development was also performed according to the OMT, and has three versions V_1, V_2, V_3 of program. Table 4 shows data which are collected from the project.

6.2.2 Effort Estimation for $\mathbf{E}_{class}(C)$

In this case, the number of versions is only 3, and is so small. Thus we take notice of classes rather than versions and investigate, for each class, the correlation between the amounts of efforts obtained by proposed method and the difficulty assigned by interview with the developers.

Thus we select 20 classes, each of which is included in the first version V_1 and is actually updated in V_2 and V_3 (We should note that, by this selection, we delete from the analysis

Table 4. Data from C++ project

Version	# of classes	LOC	Person-Days
V_1	43	6295	60
V_2	53	7765	39
V_3	63	8925	30

all classes that are in V_1 but are not updated afterwards). Then we calculate the updating effort $\mathbf{E}_{class}(C_i)$ (See formula (2)) for classes $C_1, \cdots, C_{20}$. The calculated result is shown in Table 5.

Table 5. Efforts estimation for class

Class	$E_{class}(C)$	Class	$E_{class}(C)$
C_1	23.5	C_{11}	8.3
C_2	41.7	C_{12}	11.0
C_3	16.6	C_{13}	8.6
C_4	31.7	C_{14}	5.7
C_5	9.6	C_{15}	3.3
C_6	12.4	C_{16}	5.9
C_7	8.6	C_{17}	9.5
C_8	5.8	C_{18}	3.0
C_9	7.6	C_{19}	4.9
C_{10}	8.4	C_{20}	13.7

6.2.3 Statistical Analysis

First, we had an interview with developers of target project in Experiment 2, and asked them to rank all the classes in V_1 according to the difficulty to modify. Table 6 shows the ranking of difficulty for each modified class.

From this table, C_1 is the most difficult class to update and C_2 is second. C_5 and C_6 are the same level. Finally, C_{20} is the easiest class.

Then we perform the rank correlation analysis for the data in Table 5 and Table 6. First, we make a null hypothesis H_0: there is no correlation between the rank of difficulty and the calculated efforts. As the result of analysis, the Spearman's rank correlation coefficient between them was 0.63. This result indicates that we can reject the null hypothesis H_0(The level of significance was chosen as 0.5%). Hence, we can say that the proposed weighting formulas can represent the difficulty of each class with respect to updating at a certain extent.

Table 6. Rank of difficulty

Class	Rank	Class	Rank
C_1	1	C_{11}	11
C_2	2	C_{12}	12
C_3	3	C_{13}	12
C_4	4	C_{14}	14
C_5	5	C_{15}	15
C_6	5	C_{16}	16
C_7	7	C_{17}	17
C_8	8	C_{18}	18
C_9	9	C_{19}	19
C_{10}	10	C_{20}	20

7 Conclusion

We have proposed a new cost estimation method which tries to implement intuitive, quick and easy calculation. The proposed method is designed based on the characteristics of the OO prototype development. The two experimental evaluations show that the proposed formulas and the sample weights have a certain extent of validity.

Our future works include the following:

(1) More considerations on the weight assignment are needed. Especially, theoretical basement for weight must be established or at least some empirical data are needed to validate it.

(2) We have to apply the proposed method to much more software development projects. If possible, we should collect practical project data and apply to the collected data.

References

[1] A. J. Albrecht and J. E. Gaffney: "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions of Software Engineering, Vol.9, No.6, pp.639–648, 1983.

[2] B. W. Boehm: *Software Engineering Economics*, Prentice-Hall, 1981.

[3] G. Booch: *Object Oriented Analysis and Design With Applications*, The Benjamin/Cummings, 1994.

[4] L. C. Briand, J. W. Daly and J. K. Wüst: "A Unified Framework for Coupling Measurement in Object-Oriented Systems," IEEE Transactions on Software Engineering, Vol.25, No.1, pp.91–121, 1999.

[5] 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.

[6] E. M. Kim: “Program Complexity Metric and Safety Verification Method for Object-oriented Software Development,” PhD. Dissertation, Osaka University, January, 1997.

[7] S. Kusumoto, O. Mizuno, Y. Hirayama, T. Kikuno, Y. Takagi and K. Sakamoto: “A New Project Simulator Based on Generalized Stochastic Petri-Net,” Proc. 19th International Conference on Software Engineering, pp.293–303, 1997.

[8] W. Li and S. Henry: “Object-oriented Metrics That Predict Maintainability,” Journal of Systems and Software, Vol.23, pp.111–122, 1993.

[9] M. Lorenz and J. Kidd: *Object Oriented Software Metrics*, Prentice Hall, 1994.

[10] J. Martin: *Rapid Application Development*, Macmillan Publishing Company, 1991.

[11] O. Mizuno, T. Kikuno, K. Inagaki, Y. Takagi and K. Sakamoto : “Analyzing Effects of Cost Estimation Accuracy on Quality and Productivity,” Proc. of 20th International Conference on Software Engineering, pp.410–419, 1998.

[12] J. Rumbaugh: *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

[13] I. Sommerville: *Software Engineering*, Addison-Wesley, 1992.

[14] S. Uehara, O. Mizuno, Y. Itou and T. Kikuno: “An MVC-based analysis of object-oriented system prototyping for banking related GUI applications – Correlationship between OO metrics and efforts for requirement change –,” Proc. of 4th International Workshop on Object-Oriented Real-time Dependable Systems, pp.91–104, 1999.