PAPER

A New Approach to Estimate Effort to Update Object-Oriented Programs in Incremental Development

Satoru UEHARA[†,††a)], ***Nonmember***, **Osamu MIZUNO**[††b)], ***and*** **Tohru KIKUNO**[††c)], ***Members***

SUMMARY In this paper we discuss the estimation of effort needed to update program codes according to given design specification changes. In the Object-Oriented incremental development(OOID), the requirement changes occur frequently and regularly. When a requirement change occurs, a design specification is changed accordingly. Then a program code is updated for given design specification change. In order to construct the development plan dynamically, a simple and fast estimation method of efforts for code updating is strongly required by both developers and managers. However, existing estimation methods cannot be applied to the OOID.

We therefore try to propose a straightforward approach to estimate effort for code updating, which reflects the specific properties of the OOID. We list up following factors of the effort estimation for OOID: (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), (3) the degree of information hiding is classified into private, protected and public, and (4) the degree of inheritance affects updating efforts.

We then propose a new formula $E(P, \sigma)$ to calculate the efforts needed to update a program P according to a set of design specification changes σ. The formula $E(P, \sigma)$ includes weighting parameters: W_{upd}, W_{type}, $W_{inf\text{-}h}$ and W_{inht} according to the characteristics (1), (2), (3) and (4), respectively. Finally, we conduct experimental evaluations by applying the formula $E(P, \sigma)$ to actual project data in a certain company. The evaluation results statistically showed the validity of the proposed approach to some extent.

key words: *Object-Oriented development, Incremental development process, Code updating, Effort estimation*

1. Introduction

For the software development, a lot of development paradigm has been proposed. Among them, the Object-Oriented (shortly, OO) development has high capability that transforms the real complex things into software products using the concept of object. The environments for the 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], [14]. As a result, the OO development has been used widely in industries.

It is also said that the OO development is reasonable and natural to be combined with the incremental development[8]. Because the OO paradigm makes it easy to understand the system structure and to reuse the previous components of other systems. We call the incremental development combined with the OO development as the Object-Oriented incremental development(shortly, the OOID).

In the OOID, 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 according to the new requirement. It is thus very important from the management point of view to estimate the effort for updating needed by each requirement change.

Based on the accurate estimation of successive updating activity, the managers can make an adequate re-planning of current development. So, managers have to know how much effort are needed for their updating activities. In this paper, we try to estimate the effort for updating activities using the only data that can be obtained before coding activities start.

There are many models for the effort estimation in software development. Among them, analytic models such as COCOMO model[2] and the function point analysis [1], and process models such as STATEMATE system[6] are most well-known models[13]. Analytic models consist of mathematical relationships between selected model variables, and they support management decision making with respect to the project planning. On the other hand, process models support process improvement by providing operational guidance. They are also utilized to estimate various features related to the effort, durations and the quality of final product using a stochastic simulation.

There are, however, few or no methods which take the property of the OO development into account. Furthermore, since the existing methods need a lot of preparation for customization of methods to each individual environment, they are difficult to be applied to such a small incremental development that must be done under the restrictive cost limitations.

Our objective is to develop an intuitive effort estimation method for program code updating, which is easily and cheaply applicable to the real development environment when design specification changes are derived from the requirement changes. In the method,

Manuscript received May 16, 2000.
Manuscript revised March 19, 2001.

[†]The author is with NTT Data Corporation.
[††]The authors are with the Graduate School of Engineering Science, Osaka University.
a) E-mail: ueharast@nttdata.co.jp
b) E-mail: o-mizuno@ics.es.osaka-u.ac.jp
c) E-mail: kikuno@ics.es.osaka-u.ac.jp

we propose a formula $E(P, \sigma)$ for the effort estimation where P is a certain version of program code to be updated and σ is a set of design specification changes. The $E(P, \sigma)$ is calculated as the sum of scores for all activities included in an updating.

In order to define the code updating efforts $E(P, \sigma)$, we have analyzed the activities for updating a program. First, we clarify the following 4 viewpoints to be considered in the effort estimation: (1) the kind of updating activities, (2) the types of targets to be updated, (3) the degree of information hiding and (4) the degree of inheritance. Then, we define the formula $E(P, \sigma)$ using four weighting parameters: W_{upd}, W_{type}, $W_{inf\text{-}h}$ and W_{inht} introduced to consider (1), (2), (3) and (4), respectively.

Finally, we perform experimental evaluation of our proposed method. We apply the formula $E(P, \sigma)$ 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 effort 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 OOID

2.1 Incremental development

Figure 1 shows an outline of typical incremental development process[12]. Generally speaking, the incremental development process consists of rapid interactions between customers and developers. One of specific characteristics of the incremental development is that any complete or fixed requirement do not exist before or even during the development. The developers design and implement the product based on the customers' requirements, and deliver a new version of program to the customers. Then, the customers test the new version, and return the changes of requirements to the developers. The development is iterated until the customers satisfy the product.

Generally speaking, it is very reasonable and natural to combine the incremental development and the OO development. Since the OO 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 OO development. This kind of speed-up provides a good advantage to incremental development.

2.2 Our objective

Our objective is to estimate the efforts for updating program codes in the OOID. Since the requirement changes occur frequently and regularly, the effort estimation for updating activities should also be done very fast and be easy to apply. The accurate effort estimation helps to modify development plans effectively. So, the developers in the company want to use such an estimating method that is intuitively constructed and is easy to apply.

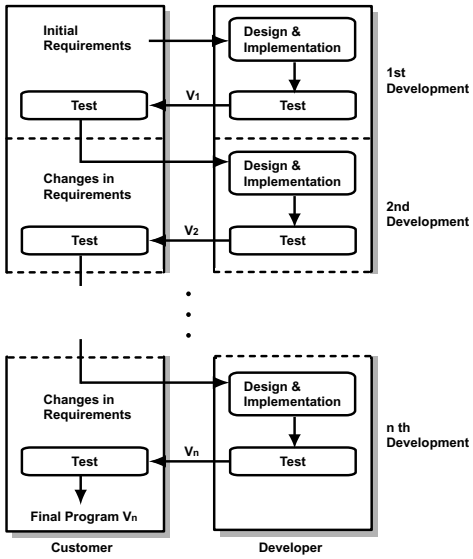


Fig. 1 Incremental development process

In the OOID, design specification changes are derived manually by the developers from given requirement changes. Of course, such activities take efforts and they are not neglectable, but the estimation of such effort is very difficult. In this paper, we do not discuss such effort.

However, the activity to update program codes from given design changes is also difficult. We therefore try to develop a effort estimation method for updating program codes from given design specification changes, which has the following properties: (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. By these properties, our proposed method can help the planning of development schedule for the next version of the software.

2.3 Related works

In order to estimate the effort for software development, many works have already been proposed. Analytic models and process models are most widely known models[13]. Our approach is included in the analytic model.

Kellner developed a process model based on the

STATEMATE system[6] to support several management control and planning activities including development schedules, planning and task re-planning by the deterministic and stochastic simulations. Kusumoto et al. also developed a simulator to estimate the efforts and the number of faults of the software project based on the generalized stochastic Petri-net[9]. These process models mainly focused on understanding and controlling the software process, and dealt with changes in the software process. However, since our objective is to estimate efforts for updating using a given design specification, it is not appropriate to apply them to our purpose.

As for the analytic model, 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. The main focus in COCOMO is upon estimating the influence of 15 cost drivers on the development effort. However, these 15 drivers do not include OO features.

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. It is thus very similar to the function point analysis (FPA)[1] in that sense. Within the FPA, the software is characterized by the five functions with respect to the number of inputs/outputs or the number of screens. These functions, however, do not include OO features. It is thus difficult to apply the FPA to OO development directly.

We therefore have to establish a new straightforward (that is, intuitive, quick and easy) method to estimate the efforts of program updating for the OOID.

3. Essential Characteristics

In order to estimate the efforts to update the program in the OOID, we chose the following 4 factors to be considered in the effort estimation: (1) the kind of updating activities, (2) the types of targets to be updated, (3) the degree of information hiding and (4) the degree of inheritance. The reasons why we chose those 4 factors will be explained in the following subsections.

3.1 Updating activities

The type of updating activity is considered to be influential element with the effort estimation because, for example, the efforts for "creating a new class," and "modifying an existing class" are clearly different in general. In the OOID, 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 a 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 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 Targets to be updated

Generally, the targets(that is, the attribute or the method in a class in this subsection) to be updated have types, and the effort to update them are different according to their types (For example, modifying a method with `int` type will take less efforts than modifying a method with a user defined 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 Object-Oriented paradigm

The Object-Oriented paradigm has several useful characteristics for high quality software development. They are encapsulation, inheritance, polymorphism, multiple inheritance and so on. Among them, we consider encapsulation and inheritance in the formulas for effort estimation.

However, we don't take account of the other properties such as polymorphism and multiple inheritance in this paper. (When a method has several interfaces by polymorphism, these interfaces can be translated into several independent methods. That's why it is considered not to be influential with the effort to update programs. Similarly, since a multiple inheritance can be

translated into plural inheritances, we treat the multiple inheritance as plural inheritances and count the effort for each inheritance.)

3.3.1 Degree of information hiding

An aspect of the encapsulation is measured by the degree of information hiding. The degree of information hiding by encapsulation clearly affects the updating efforts, since the scopes of variables are different according to the degree of information hiding. 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.

For example, suppose that a class has some "public" attributes. When modifying this class, it should be checked whether public attributes are referred from the other classes. Then more effort is needed to modify this class.

3.3.2 Degree of inheritance

In the Object-Oriented development, programmers can create a new class which inherits properties of existing classes. The aspect of inheritance contributes to a high productivity and helps reuse of previous components in the other programs or class libraries.

On the other hand, the degree of inheritance is influential with the updating efforts, since programmers must pay attention to the children classes when they update the parent class. In the formulas for the efforts estimation, we assign each class distinct score which is proportional to the number of children classes.

4. Formula for Effort Estimation

4.1 Outline of formula

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

Assume that P is a program to be updated according to design specification changes σ. In more precise, σ is a set of n design specification changes $SC_1, \cdots, SC_n$. Then P' is a resultant program that is obtained from P by executing necessary operations specified in $SC_i (1 \leq i \leq n)$. In this paper, we propose the following formula to estimate the efforts for updating program code P:

$$\begin{aligned} E(P,\sigma) &= \sum_{i=1}^{n} E_{req}(SC_i) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m_i} E_{cl}(C_j^i) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m_i}\Big(\sum_{k=1}^{p_{ij}} E_{attr}(A_k^{ij}) + \sum_{l=1}^{q_{ij}} E_{meth}(M_l^{ij})\Big) \end{aligned} \tag{1}$$

Intuitively speaking, according to design specification change SC_i, a class C_j^i is updated. In more detail, an attribute A_k^{ij} in the class C_j^i and a method M_l^{ij} in the class C_j^i are updated. We will explain the definition of E's in the following sections. At this point, we cannot validate this formulation theoretically. Currently, Equation (1) is based on the interviews with the actual developers in a certain company.

4.2 Four kinds of efforts

1) Efforts to meet a design specification change SC
   Assume that m classes $C_1, \cdots, C_m$ are updated for the design specification change SC. The formula to estimate the efforts for design specification change SC is defined as follows:

$$E_{req}(SC) = \sum_{j=1}^{m} E_{cl}(C_j) \tag{2}$$

   We explain the definition of $E_{cl}(C_j)$ in the following.
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:

$$\begin{aligned} &E_{cl}(C) \\ &= \sum_{k=1}^{p} E_{attr}(A_k) + \sum_{l=1}^{q} E_{meth}(M_l) \end{aligned} \tag{3}$$

   We explain the definition of $E_{attr}(A_k)$ and $E_{meth}(M_l)$ in the following.
3) 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:

$$\begin{aligned} &E_{attr}(A) \\ &= \alpha \times W_{upd} \times W_{type} \times W_{inf\text{-}h} \end{aligned} \tag{4}$$

   The semantics of variables are summarized as follows:

   a) α: 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).

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

$$\begin{aligned} & E_{meth}(M) \\ & = \beta \times (1 + WCC(M) + WCM(M) \\ & \quad + WPM(M) + W_{inht} \times NOC(C)) \\ & \quad \times W_{upd} \times W_{type} \times W_{inf\text{-}h} \end{aligned} \tag{5}$$

where variables β, W_{upd}, W_{type}, $W_{inf\text{-}h}$, W_{inht} are constants to represent characteristics in Section 3 and $WCC(M)$, $WCM(M)$, $WPM(M)$, $NOC(C)$ are fundamental OO metrics to be defined in the next subsection. In Equation (5), metrics $WCC(M)$, $WCM(M)$, $WPM(M)$ and $NOC(C)$ are all added rather than multiplied, since they represent internal properties in a method M and they are independent from their definition. Furthermore, from the interviews with developers in a certain company, the efforts that correspond to these metrics are considered to be independent, too.
The semantics of these variables are summarized as follows:

a) β: 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).
e) W_{inht}: a basic weight representing the difference in the difficulty caused by the degree of inheritance.

When we apply the proposed formula to actual development, we have to determine the values of α, W_{upd}, W_{type}, $W_{inf\text{-}h}$ in the formula (4) and β, W_{upd}, W_{type}, $W_{inf\text{-}h}$, W_{inht} in the formula (5). 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], [7], [10], [11], most of them are for a class, but not for a method. So, we suggest simple new metrics WCC, WCM, WPM and NOC which indicate the 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 w_{C_i} for a class C_i, and then define WCC as follows:

$$WCC(M) = \sum_{i=1}^{n} w_{C_i}$$

In fact, the formula of WCC is a part of the definition of existing OO metric 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 the same class C. We introduce the weight w_{A_i} for an attribute A_i, and then define WCM as follows:

$$WCM(M) = \sum_{i=1}^{n} w_{A_i}$$

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 w_{P_i} for a parameter P_i, and then define WPM as follows:

$$WPM(M) = \sum_{i=1}^{n} w_{P_i}$$

4) NOC(Number Of Children)
NOC is a part of the definition of existing OO metrics set[5]. It is a measure of how many subclasses are going to inherit the methods of the parent class:

$$NOC \quad = \quad \begin{array}{l}\text{the number of immediate} \\ \text{subclasses subordinated to} \\ \text{a class in the class hierarchy}\end{array}$$

4.4 Limitations

There are several limitations in the proposed effort estimation.

1) Programming languages
We assume C++ and JAVA as programming languages in the proposed effort estimation. Since most of OO programs in the real field are written in C++ or JAVA, this limitation is not significant.
2) Development process
We take account of incremental development model rather than waterfall model. The main reason is that the incremental development is the mainstream in the OO development.

3) Collection of data
In order to apply our effort estimation to projects, the values of parameters must be decided before coding phase. Since the data needed for these parameters are easy to collect, there is no serious problem with respect to applicability.

5. Parameter Tuning

5.1 Parameters

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 analyzed 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 Equation (2))
(2) Attributes to be updated in each class C_i(that is, attributes $A_1, \cdots, A_p$ in Equation (3))
  2-1) The kind of updating activity
  2-2) The type of the attribute
  2-3) The degree of information hiding
  2-4) The degree of inheritance
(3) Methods to be updated in each class C_i(that is, methods $M_1, \cdots, M_q$ in Equation (3))
  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 and NOC, 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
    3-4-4) The number of subclasses

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

5.2 Target project

In order to determine the values of parameters in Equation (3), we had an experiment using the data of an actual project. In this experiment, we tuned up the values of parameters so that the rank correlation coefficient between them becomes large.

The targeted project was a development of an application software for a banking related system[16]. The program was written in C++ on Microsoft Windows 95/98/NT. The development was also performed according to the OMT[14], and had three versions V_1, V_2, V_3 of the program. Table 1 shows the data which are collected from the project. In Table 1, $V_i (i = 1, 2, 3)$ denotes the ith version of programs and "# of classes" denotes the total number of classes included in V_i. The value of LOC denotes the total lines of code excluding the comments, and the value of *Person-Days* indicates the effort needed to develop each version.

Table 1 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

We firstly selected 20 classes, each of which was included in the first version, V_1, and was actually updated in V_2 or V_3 (We should note that, by this selection, we delete all classes that were in V_1 but were not updated afterwards from the analysis).

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

From Table 2, we can see that C_1 is the most difficult class to update and C_2 is second. C_5 and C_6 are the same level, and C_{20} is the easiest class. We assumed that the most difficult class by developers' intuition needs the most effort to update it.

Table 2 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

5.3 Process of tuning up

We tuned up the values of parameters so that the rank of estimated effort becomes almost the same as the difficulty of classes in Table 2. The steps for tuning up are as follows:

Step 1 Firstly, we set initial values of parameters as shown in Table 3.
Step 2 We calculate the effort for each class according to Equation (3).

Table 3 Initial values of parameters

(a) Parameters for attributes ($\alpha = 1$)

W_{upd}	Creation	Deletion	Modifi.
	1	1	N/A
W_{type}	Basic	Library	Custom
	1	1	1
$W_{inf\text{-}h}$	Private	Protected	Public
	1	1	1

(b) Parameters for methods ($\beta = 1$)

W_{upd}	Creation	Deletion	Modifi.	
	1	1	1	
W_{type}	Void	Basic	Library	Custom
	1	1	1	1
$W_{inf\text{-}h}$	Private	Protected	Public	
	1	1	1	
W_{inht}	0.1			

(c) Parameters for software metrics

	Basic	Library	Custom
w_C	N/A	0.1	0.1
w_A	0.1	0.1	0.1
w_P	0.1	0.1	0.1

Step 3 We then compare the difficulties with obtained efforts by rank correlation analysis.

Step 4 In order to acquire better rank correlation coefficient, we change some values of parameters based on some conditions, then back to Step 2. Or if the current correlation coefficient is considered as the best one, accept the values of parameters.

The conditions used in Step 4 are summarized as follows:

- The effort needed to create an attribute is larger than or equal to that of deletion.
- Custom types need more efforts to update than library types, and library types need more than basic types.
- Public attributes or methods need more efforts to update than protected ones, and protected ones need more than private ones.
- Generally, the effort to update attribute is larger than or equal to that of to update method.
- Based on the interviews with the developers in a certain company, W_{upd} should be larger than W_{type}.
- Based on the interviews with the developers in a certain company, W_{type} should be almost the same as $W_{inf\text{-}h}$.
- Based on the interviews with the developers in a certain company, the values of w_C, w_A and w_P are considered almost the same.

(Note that these conditions are mainly based on the interviews with the actual developers in a certain company. In the further research, we have to show the validity of this work empirically.)

According to these steps, we actually determined the values of parameters. The obtained values are shown in Table 4. Table 4(a) shows values of W_{upd}, W_{type}, $W_{inf\text{-}h}$ for attribute A_i. In this case, the parameter α became 3. Table 4(b) shows values of W_{upd}, W_{type}, $W_{inf\text{-}h}$, W_{inht} for method M_i. The parameter β became 1. Table 4(c) shows values of software metrics w_C, w_A and w_P.

The estimated effort for each class, which was calculated using the values of parameters in Table 4, is shown in Table 5. The Spearman's rank correlation coefficient between estimated and actual efforts for each class became 0.81. We thus can say that the values of parameters are optimized well.

Table 4 Final values of parameters

(a) Parameters for attributes ($\alpha = 3$)

W_{upd}	Creation	Deletion	Modifi.
	10	2	N/A
W_{type}	Basic	Library	Custom
	1	2	3
$W_{inf\text{-}h}$	Private	Protected	Public
	1	2	3

(b) Parameters for methods ($\beta = 1$)

W_{upd}	Creation	Deletion	Modifi.	
	5	1	3	
W_{type}	Void	Basic	Library	Custom
	1	1	2	3
$W_{inf\text{-}h}$	Private	Protected	Public	
	1	2	3	
W_{inht}	0.1			

(c) Parameters for software metrics

	Basic	Library	Custom
w_C	N/A	0.1	0.12
w_A	0.05	0.1	0.12
w_P	0.05	0.1	0.12

Table 5 Efforts estimation for class

Class	$E_{cl}(C)$	Class	$E_{cl}(C)$
C_1	1722.5	C_{11}	159.7
C_2	593.2	C_{12}	242.2
C_3	352.9	C_{13}	129.7
C_4	2346.7	C_{14}	87.6
C_5	178.9	C_{15}	75.2
C_6	169.9	C_{16}	93.7
C_7	165.5	C_{17}	119.4
C_8	113.3	C_{18}	50.5
C_9	169.9	C_{19}	74.8
C_{10}	258.8	C_{20}	155.9

6. Experimental Evaluations

In order to evaluate the validity of the proposed method and the values of parameters shown in Table 4, we had another experiments using the other projects' data. (Another possibility may be an experiment which applies the values in Table 4 to the target project data mentioned in subsection 5.2. But since only three versions of programs were developed in the project, we cannot analyze well on this project.) In these experiments, we got the data from two development projects of OO program. We then evaluated the correlation between the efforts estimated by proposed method and actual ones for each updating activity.

6.1 Experiment A

6.1.1 Collected Java programs

In the first experiment, the targeted project was a development of an application software that shows graphical images stored in Object-oriented database[17]. The program was written in JAVA, and it was able to run on a WWW browser. The development was performed according to the OMT, and 10 versions of programs were successively developed.

Table 6 shows the data that were collected from the project. In Table 6, $V_i (i = 1, 2, \cdots, 10)$ denotes the ith version of programs and "# of classes" denotes the total number of classes included in V_i. The value of *LOC* denotes the total lines of code excluding the comments, and the value of *Person-Days* indicates the effort needed to develop each version.

Table 6 Data from a JAVA project used in experiment A

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 $E(P, \sigma)$

The values of parameters used in the estimating formula were the same as in Table 4. Since the development environment or the skill of the development team of this project were almost the same as that of in subsection 5.2, we were able to apply those values of parameters to this project.

We calculated the updating effort $E(P, \sigma)$ for each version $V_1, \cdots, V_{10}$ by Equotion (1). The result was summarized in Table 7.

Table 7 Efforts estimation for each version

Version	$E(P, \sigma)$
V_1	94271.4
V_2	15843.6
V_3	10002.0
V_4	8384.7
V_5	6662.3
V_6	9241.3
V_7	2120.7
V_8	1013.8
V_9	242.6
V_{10}	171.3

6.1.3 Statistical analysis

We then evaluated statistically these estimated efforts $E(P, \sigma)$ in Table 7 by comparing with actual *Person-Days* in Table 6.

Note that we exclude the data of V_1 (initial version) from the correlation analysis. Since our objective is to estimate the efforts for updating activity, we excluded V_1 because it is a new development of programs in the project†.

The correlation coefficient was calculated as 0.71, and it was statistically significant with 1% level of significance. Figure 2 shows the relationship between them.

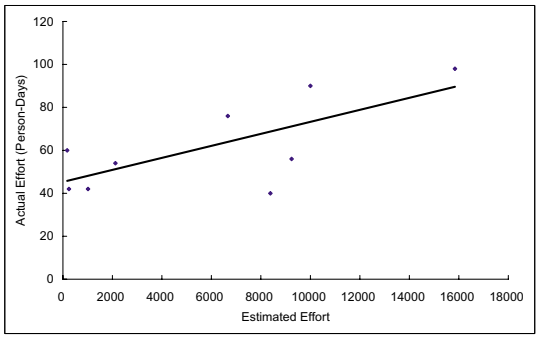


Fig. 2 Actual efforts and $E(P, \sigma)$

6.2 Experiment B

In the second experiment, the targeted project was a development of an application software that calculates software metrics in several programming languages. The program was written in C++, and the size of development was about 20 KLOC. There were 9 versions available for this experiment.

Table 8 shows the result of experiment B. In Table 8, V_i $(i = 1, 2, \cdots, 9)$ denotes the ith version of programs, the *Person-hours* indicates the effort needed to develop each version, and $E(P, \sigma)$ shows the program updating effort.

We also evaluated statistically these estimated efforts, $E(P, \sigma)$, and actual *Person-hours* in Table 8. The correlation coefficient was calculated as 0.83††, and it was statistically significant with 1% level of significance.

†Note that our estimation approach seems to be applicable logically to any versions, including an initial version, of a software development. The initial version, however, has special activities such as designing the algorithms and implementing highly reusable components, which may not be represented in the proposed formula. That's why we exclude the initial version of the development from experiments.

††We also exclude the initial version from calculation of the correlation coefficient.

Table 8 Result of experiment B

Version	*Person-hours*	$E(P,\sigma)$
V_1	335.8	19951.3
V_2	33.2	1212.5
V_3	71.8	2604.7
V_4	15.3	1213.4
V_5	105.2	2425.8
V_6	13.3	575.9
V_7	33.7	699.6
V_8	41.3	546.5
V_9	20.5	301.0

From these result of experiments A and B, we can say that there is high correlation between efforts calculated by the proposed formula and actual ones. That is, we can confirm that the effort estimation using the values of parameters in Table 4 is useful for the other development project.

7. Conclusion

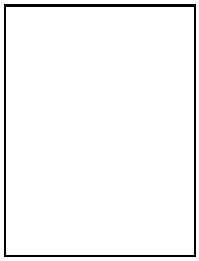
We have proposed a new effort estimation method which tries to implement intuitive, quick and easy calculation. The proposed method is designed based on the characteristics of the OOID. The two experimental evaluations show that the proposed formulas and the sample weights have a certain extent of validity.

Our future works include the following:

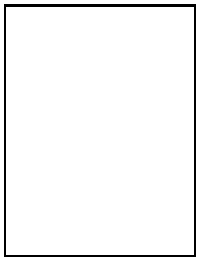
(1) More considerations on the weight assignment are needed. Especially, theoretical basement for weight must be established or at least some empirical data are needed to validate it.
(2) We have to apply the proposed method to much more software development projects. If possible, we should collect practical project data and apply the proposed method to the collected data.

References

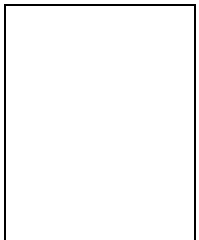
[1] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," IEEE Trans. Software Eng., vol.9, no.6, pp.639–648, 1983.
[2] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
[3] G. Booch: *Object Oriented 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 Trans. Software Eng., vol.25, no.1, pp.91–121, 1999.
[5] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Trans. Software Eng., vol.20, no.6, pp.476–493, 1994.
[6] M. I. Kellner, "Software process modeling support for management planning and control," Proc. 1st International Conference on the Software Process, pp.8–28, 1991.
[7] E. M. Kim, "Program complexity metric and safety verification method for object-oriented software development," PhD. dissertation, Osaka University, January, 1997.
[8] P. Kruchten, *The rational unified process: An introduction*, Addison-Wesley, 1999.
[9] 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.
[10] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," Journal of Systems and Software, vol.23, pp.111–122, 1993.
[11] M. Lorenz and J. Kidd, *Object Oriented Software Metrics*, Prentice Hall, 1994.
[12] J. Martin, *Rapid Application Development*, Macmillan Publishing Company, 1991.
[13] D. M. Raffo, "Evaluating the impact of process improvements quantitatively using process modeling," Proc. CASCON93, vol.1, pp.290–313, 1993.
[14] J. Rumbaugh, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.
[15] I. Sommerville, *Software Engineering*, Addison-Wesley, 1992.
[16] 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. 4th International Workshop on Object-Oriented Real-time Dependable Systems, pp.91–104, 1999.
[17] S. Uehara, O. Mizuno and T. Kikuno, "A straightforward approach to effort estimation for updating programs in object-oriented prototyping development," Proc. 6th Asia-Pacific Software Engineering Conference, pp.144–151, 1999.



Satoru Uehara was born in 1971. He received B.E. degree in information and computer sciences from Osaka University in 1993. He has been working for NTT DATA Corporation, and has been engaged in the development of client-server systems. He is currently a Ph.D course student in the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University. His research interests include software metrics, object-oriented development methodologies.



Osamu Mizuno was born in 1973. He received B.E. and M.E. degrees in information and computer sciences from Osaka University in 1996 and 1998, respectively. He has been working for Osaka University since 1999. He is currently a Research associate in the Department of Informatics and Mathematical Science at Osaka University. His research interests include the software process and the software quality assurance technique. He is a member of the IEEE.



Tohru Kikuno was born in 1947. He received M.S. and Ph.D. degrees from

Osaka University in 1972 and 1975, respectively. He joined Hiroshima University from 1975 to 1987. Since 1990, he has been a Professor in the Department of Informatics and Mathematical Science at Osaka University. His research interests include the quantitative evaluation of software development processes and the analysis and design of fault-tolerant systems. He served as a program co-chair of the 1st International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98) and of the 5th International Conference on Real-Time Computing Systems and Applications (RTCSA'98). He also served as a general co-chair of the 2nd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '99).