A Verification of Class Structure Evolution Model and its Parameters

Mikio OHKI
Nippon Institute of Technology
4-1 Gakusendai Miyashiro
Saitama Japan
+81-480-33-7466
E-mail: ohki@nit.ac.jp

Shinjiro AKIYAMA
JIPEngineering Service Co.,Ltd
8-3 Koamicho Nihonbashi Chuoku
Tokyo Japan
+81-3-3808-1361

Yasushi KAMBAYASHI
Nippon Institute of Technology
4-1 Gakusendai Miyashiro
Saitama Japan
+81-480-33-7466
E-mail: yasushi@nit.ac.jp

ABSTRACT

It is widely accepted that the role of software “architect” that provide frameworks to program developers is important in the object-oriented software development processes. When developers try to extend the base classes given by the architect, they may want some guidelines that tell them how many subclasses and how many methods in one subclass are reasonable. So far we are not aware of such guidelines. Through measurements of Java and Delphi class libraries, we have distilled formulae that forecast the number of methods and the number of subclasses when constructing class trees from the base classes. We propose that we should focus to extract methods and attributes rather than class structure. The formulae we have formulated support this proposition.

Categories and Subject Descriptors

[**Metrics**]: *Object Oriented Program, Software Evolution.*

General Terms

Measurement, Experimentation, Verification.

Keywords

Evolution model, Architect, Measurement, Verification.

1. INTRODUCTION

It is well known that using pre-organized class libraries is effective in the object-oriented software development processes. Jacobson et al. have proclaimed that the key person in such software development is the “architect” who defines the framework for the target application [1].

The major task of the architect is to prepare the base classes for the application domain so that the fellow developers can use these classes as the framework for the application. Although architects usually provide information about class structure as well as the function and usage of each class, they do not provide how to extend those base classes through inheritance and composition. What the developers would like to know is the guidelines which of the following options and when they should employ them to construct the subclasses. The options they can choose are:

(1) Construct one subclass and pack all the methods in it.

(2) Construct several subclasses in the same level, immediately one under the super class, and make each subclass have a group of methods.

(3) Construct yet another structured set of subclasses which has hierarchical structure.

So far we are not aware of rules that can be used as guidelines to construct the complete class structure based on a given application framework. Of course, we have some empirical knowledge rules such as “Class hierarchy should be based on ‘cases’,” but we would like to have quantitative guidelines such as how many subclasses should be constructed under one super class and how many methods each subclass should have.

We have found that well-organized class libraries have some common structural pattern in their class hierarchy, and that such patterns are preserved through class evolution. Therefore we have analyzed the statistical characteristics of well-organized class libraries, and distilled such patterns. The patterns suggest a good way to construct subclasses of application framework.

In this paper, we describe: 1) our hypothesis that the structure of a class is the history of the class evolution, 2) the idea to formulate model formulae to construct subclass, and 3) parameters for the formulae statistically computed from class libraries of Java and Delphi. We close our discussion with a new hypothesis that is suggested those formulae and the verification of them. We ignored the “interfaces” in Java to simplify the discussion.

2. THE CLASS EVOLUATION HIPOTHESIS

Several studies have tried to quantify to what extend the number of methods and attributes are correlated with class structure [2]. Nakatani et al. have suggested, “Inheritance is a means to adapt to a new circumstance caused by requests for changes that the super class can not handle” [3]. This thesis is based on the hypothesis that the class structure shows the history of the given application. Reading class structure, we can trace how the application has adopted to the new requirements and how each class has survived in the course of design selections. Our discussion is based on the idea that such effort for adoption is the driving force of the inheritance.

The first step toward the guideline for subclass construction is to find the relationship between a super class and the immediate subclasses. Through such relation, we can statistically forecast the number of subclasses, attributes and methods. In order to analyze class libraries, we use two viewpoints as follows:

1) The relationship between the characteristics of a super class level *i*, i.e. the number of attributes λ_i and the number of methods ξ_i, and those of the subclasses level *i+1*, i.e. the number of attributes λ_{i+1} and the number of methods ξ_{i+1}.

2) The relationship between the characteristics of a super class level *i* and the number of the subclasses of the super class n_{i+1}.

The observation upon the class libraries of Java and Delphi has suggested that the number of methods and the number of attributes in all the subclasses with level *i+1* are related to the number of methods and the number of attributes in the common super class level *i*, respectively. These relationships can be expressed as the following formulae.

$$\Sigma \lambda_{i+1} = f(\lambda_i) \qquad \cdots\cdots (1)$$

$$\Sigma \xi_{i+1} = g(\xi_i) \qquad \cdots\cdots (2)$$

There are two types of methods in the subclasses. One group is a set of new methods with new names that simply add new functions to the subclasses. The other is a set of methods that finalize the inheritance chain so that the subclasses of the subclass cannot inherit those methods (by using keywords “private” and “final”). Therefore the formula (2) can be refined as formula (3). In this formula, $\alpha \xi_i$ expresses the increasing factor for the number of methods of the first group proportional to the number of methods in the super class ξ_i, and $(1-\beta \xi_i)$ expresses the decreasing factor for the number of methods of the second group proportional to the number of methods in the super class. The increasing factor α stands for the growth rate of the number of newly added methods in subclasses. The decreasing factor β stands for the ratio of the methods that finalize the inheritance.

$$\Sigma \xi_{i+1} = \alpha \xi_i (1-\beta \xi_i) \qquad \cdots\cdots (3)$$

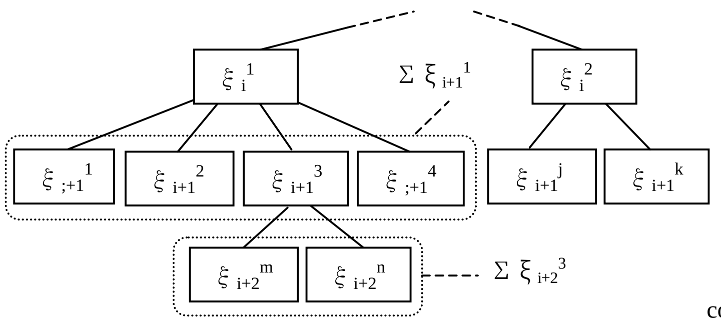

Figure 1. The number of methods in each class

The formula(3) describes that the number of methods in all the subclasses, $\Sigma \xi_{i+1,}$ is determined by the cross-correlation between the increasing factor and the decreasing factor. The number of methods in all the subclass is expressed by a quadratic equation. In other words, it represents a logistic-like mapping function that the number of the methods in all the subclasses with level *i+1* and the number of methods in the super class level *i*, transit themselves with keeping the autocorrelation-ship. This situation is depicted in Figure 1.

Unlike the case of methods, the number of attributes in all the subclasses increases monotonically. Therefore, the relationship between the number of attributes in a super class and the number of all the subclasses can be conjectured as follows:

$$\Sigma \lambda_{i+1} = \gamma \lambda_i + \delta \qquad \cdots\cdots (4)$$

The number of subclasses of the super class level *i* can be conjectured as follows:

$$n_{i+1} = \varepsilon \xi_i^{-\theta} \qquad \cdots\cdots (5)$$

The formula (5) describes that a super class with many methods has a small number of subclasses, and a super class with few methods has many subclasses. Since the inheritance is based on “cases,” it is reasonable that a super class with much functionality has fewer subclasses than that with little functionality.

3. VERIFICATION OF HIPOTHESIS

3.1 Two Groups of Relations

In order to verify the hypotheses, we have counted the methods of classes in ComponentUI in Java class library. The relationship between all the classes and their subclasses is shown in Figure 2.

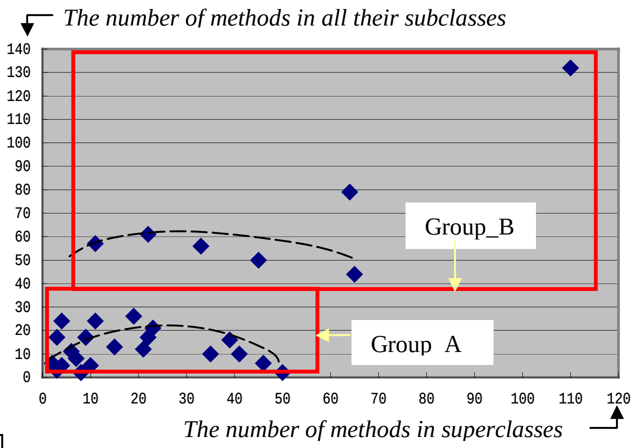

Figure 2. The relationship between all the classes and their subclasses

Figure 2 displays two distinct groups of relations. We conjectured that the relationship between a super class that is near the root of a class hierarchy and its subclasses may be different from the relationship between a super class that is near the leaf of a class hierarchy. Therefore we divided classes into two groups A and B based on the number of methods, and observed where those classes in each group are found in the class hierarchy. Group A consists of classes with the number of methods less than thirty, and group B consists of classes with the number of methods more than or equal to thirty.

The number of “thirty” is chosen heuristically. We compared the average distance of each class from the leaf of class structure. The results are shown in Table 1. We employed the t-test and found statistical significance (The null hypothesis was rejected with 5% critical value.) Classes of group A reside closer to the root of class hierarchy than classes of group B. Classes of group B reside relatively close to the leaf classes.

Table 1. Comparison of the distances from the leaves between group A and group B

	Group A	Group B
Number of Classes	25	7
Average Distance from Leaf (Number of levels)	1.292	2.000
Variance of Average Distance	0.373	0.285
Standard Deviation of Average Distance	0.624	0.577
Computed t-value	-2.61	

Classes of group A behave according to the formula (3), but classes of group B behave differently. It seems that classes of group B have linear relation with respect to super classes and their subclasses. Therefore, the relation between the number of methods in all the subclasses and the number of methods in their super class of group B can be expressed in a linear formula as follows:

$$\Sigma\ \xi_{i+1} = a\,\xi_i + b \qquad \cdots\cdots(6)$$

Upon these observation, we determined to find parameters α and β for group A, and parameters *a* and *b* for group B.

3.2 Estimating Parameters for Formulae

(a) Java Class Library

Since “ComponentUI” and “Component” in Java class library provide enough classes for measurement, we employed the least squares method to obtain parameters α and β, and *a* and *b* for formulae(3) and (6), respectively. The results are shown in Table 2. The correlations among group A are shown in Figures 3 and 4. Classes of group B show linear correlations.

(b) Delphi Class Library

We performed the similar analysis against VCL (Visual Component Library) of Delphi. We chose four class trees, TObject, TPersistant, and TWinControl, because of their rich class hierarchy. Since most of the classes have more than thirty methods, i.e. group A methods, we performed the regression analysis to obtain α and β for formula (3). The results are shown in Table 3.

Table 2. Parameters for forecasting formulae computed from Java class library

NOCT	NOC	Forecasting Model Formula	Parameters	PCP	LOS
ComponentUI					
Group_A	24	Quadratic	$\alpha = 1.534$ $\beta = 0.020$	0.54	5%
Group_B	7	Linear	$a = 1.059$ $b = 8.367$	0.76	5%
Component					
Group_A	14	Quadratic	$\alpha = 1.874$ $\beta = 0.028$	0.47	5%
Group_B	10	Linear	$a = 1.305$ $b = 15.683$	0.76	5%

NOCT: Names of class tree
NOC: Number of sample classes
PCP: Pearson correlation parameters
LOS: Level of significance
/The level of significance represents the reject level for the null hypothesis of the Pearson correlation parameters. /

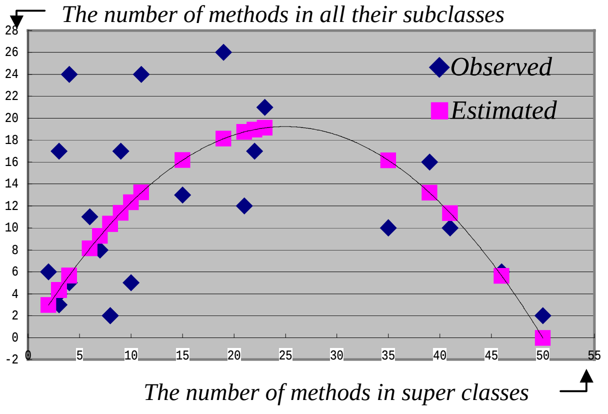

Figure 3. Relation between the number of methods in super classes and the number of methods in all their subclasses found in group A of ComponentUI

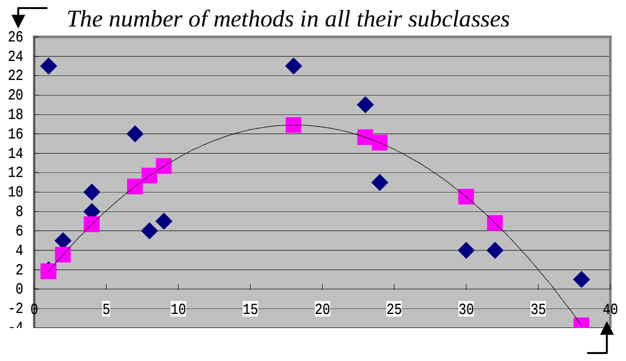

Figure 4. Relation between the number of methods in super classes and the number of methods in all their subclasses found in group A of Component

Table 3. Parameters for forecasting formulae computed from VCL of Delphi

NOCT	NOC	Forecasting Model Formula	Parameters	PCP	LOS
TObject	12	Quadratic	$\alpha = 2.313$ $\beta = 0.017$	0.86	1%
TPersistant	6	Quadratic	$\alpha = 6.152$ $\beta = 0.019$	0.74	10%
TComponen	5	Quadratic	$\alpha = 4.215$ $\beta = 0.065$	0.93	1%
TWinControl	11	Quadratic	$\alpha = 2.921$ $\beta = 0.009$	0.63	5%

3.3 Forecasting Formulae for the Number of Subclasses

Next, we find the parameters in formula (5), i.e. ε and θ. The relationship between the number of methods in a class and the number of methods in all the subclasses can be depicted in Figure 5.

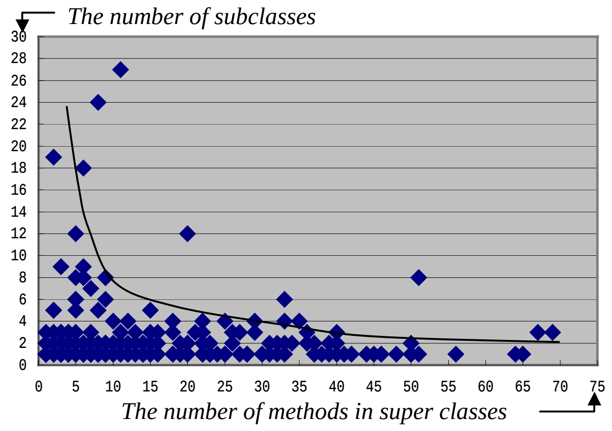

Figure 5. Relation between the number of methods in super classes and the number of their subclasses

Upon applying regression analysis to these data, we obtained the forecasting model formula(7) as follows(the number of samples is fifty, the correlation parameter is 0.60). This formula forecasts the maximum number of subclasses of a super class. We will scrutinize this formula in Section4.

$$n_{i+1} = 18.15\, \xi_i^{-0.584} \quad \cdots\cdots (7)$$

4. OBSERVATIONS AND DISCUSSION

4.1 Rationale of the Formula for the Number of Methods

In the previous section, we verified the formula (3), the number of methods in all the subclasses is determined by the cross-correlation between the increasing factor and the decreasing factor, by using the Java class libraries, i.e. ComponentUI and Component, and VCL of Delphi. The observed data show statistical significance. From the observation, we can conclude that the number of methods in all the subclasses is determined by the cross-correlation between the increasing factor and the decreasing factor. We believe that the software evolution appears as the increase of the number of methods in the target software. We demonstrate such increase of the number of methods can be expressed by the logistic-like mapping function. We can say that the class libraries that we used for verification have evolved along the model formulae that we developed.

4.2 Rationale of the Formula for the Number of Subclasses

One way to explain the fact that the number of subclasses is inverse proportional to the power of the number of methods in the immediate super class is introducing a new concept, namely the connecting force of methods. Such conceptual force among methods can be formulated as follows:

$$F = C\, m^{\theta} \quad \cdots\cdots (8)$$

In this formula, *C* and *m* stand for a constant and the number of methods in a class, respectively. For example, when a method in a class has interactions with all methods in the class including the method itself, the number of the interactions is m^2. If we assume such connecting force, constructing a subclass requires another imaginary force to extract methods from the super class against this connecting force. Therefore, even though the requirement for subclasses occurs in a constant probability, the frequency that the requirement is satisfied with a certain effort is inverse proportional to the connecting force. Upon applying this hypothesis to the number of subclasses and the number of methods in the super class, it is easy to understand the fact that the maximum number of subclasses of a super class is inverse proportional to the number of methods in the super class. This hypothesis explains the formula (7).

4.3 Class Modeling Based on Attributes

It is demonstrated that the number of methods in all the subclasses is expressed in logistic-like mapping function. The logistic mapping function is known that it can be used to forecast the variation of population. This fact suggests that methods may determine the characteristics of the class. In other words, one should construct a class from methods in bottom-up way. We would like to propose the following propositions for discussions.

(1) The task of a method is modifying some attributes. Through this modification, a method affects the behavior of other methods. We should pay more attention to methods and attributes rather than classes. Classes can be seen as mere containers of methods and attributes. When we design software, we should extract methods and attributes before constructing class structures. The methodology that CRC cards employs suggests the same approach [5][6]

(2) There should be some rules that we can use for constructing classes from methods (and attributes.) The other study of ours on the timing of data generation and method implementation suggests these constructing rules [7]. Formulating these rules is the further research direction.

5. ACKNOWLEDGMENTS

A part of this research was performed as one of the HITOCC projects and partially sponsored by Japan Information Technology Services Industry Association.

6. REFERENCES

[1] I.Jacobson,G. Booch,J. Rumbaugh,"The Unifield Software Development Process," Addison Wesley (1999)

[2] Chidamber＆Kamemer,"A Metrics Suite for Object Oriented Design,"IEEE Trans. SE Vol.20, No.6 pp.476-493 (1994)

[3] Takako Nakatani, Tetuo Tamai, "A Study on Statistic Characteristics of Inheritance Tree Evolution," Proceedings of Object-Oriented Symposium, IPSJ, pp.137～144（1999）. In Japanese.

[4] Mikio Ohki,Shoijiro Akiyama,"A Class Structure Evolutional Model and Analysis of its Parameters," IPSJ Vol.2001 No.92 SE-133-3 pp.15-22(2001) . In Japanese.

[5].Wirfs-Brock, B.Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach," Proc of OOPSLA'89, ACM, pp. 71-75, 1989.

[6].Wirfs-Brock,"Designing Objects and Their Interactions: A Brief Look at Responsibility-Driven Design," Carroll, J. M. ed., Scenario-Based Design, John Wiley & Sons,1995

[7] Mikio Ohki,Kohei Akiyama," A Proposal of the Conceptual Modeling Criteria and their Validity Evaluation ," IEICE VOL.J84-D-1 No.6 pp.723-735(2001)