Evolutional Characteristic of Class Inheritance Trees

Takako NAKATANI Tetsuo TAMAI
Graduate School of Arts and Sciences
University of Tokyo
3-8-1, Komaba, Meguro-ku, Tokyo 153, Japan
Tel: +81-3-5454-6847
tina@graco.c.u-tokyo.ac.jp, tamai@graco.c.u-tokyo.ac.jp

ABSTRACT
In our previous work, we studied object evolution processes by tracing a real case of software development. It was shown that class inheritance trees tend to keep their peculiar values of lines of code per method. The peculiar value can be considered to represent a characteristic of an inheritance tree and to affect class evolution as a constraint.

We continued the work and have studied other two software development cases. The objectives are to examine whether the existence of tree peculiar values is a peculiar phenomenon of the first case, whether the values affect class evolution directions, and whether the values are derived from developers' design habits. Our conclusions are that class trees have their peculiar values independent of developers and classes of a tree tend to have a unique common value determined by the tree properties.

Keywords
object-oriented software, metrics, evolution, inheritance tree

INTRODUCTION

We study several software development cases to reveal object evolution processes quantitatively. Purposes of our studies are:

1. to evaluate object maturity for reuse quantitatively by analyzing object change processes,
2. to indicate time to redesign classes, and
3. to show object design properness by statistical data.

In our previous work, the first purpose was pursued by observing object evolution processes. As a result, several different types of object evolution processes were revealed. One type of objects become stable after changing their specifications as the system grows. Such objects would be treated as reusable classes after reaching the stable stage[8].

Statistical data such as the number of lines of code per class and the number of methods per class, measured over the whole system, have non-Gaussian distribution[1, 6]. The distribution pattern has its median on the left side and long tail on the right side. To show the class design properness and time to redesign classes, we have traced the paths of classes that had been moving in the measured space[7]. Some classes changed their statistical values frequently and generally tended to increase their values. If the designers had watched class moving paths through measurement, they could have objectively extracted classes needed to be redesigned. Therefore, tracing object evolution processes is useful to evaluate design properness.

We study to pursue the third purpose by focusing on tree evolution processes and to clarify evolutional characteristic of class inheritance trees.

This paper is constructed by the following sections. The next section introduces the results of our previous study and proposes hypotheses. The following section shows other software development case studies to verify the hypotheses. The remainder of this paper discusses the results of the verification.

PRELIMINARY EXAMINATION

Applied Metrics

To evaluate the degree of object changes, we use metrics called CK metrics proposed by S. R. Chidamber and C. F. Kemerer as object-oriented design metrics[3]. We adapt WMC (Weighted Methods per Class) and DIT (Depth of Inheritance Tree) defined in CK metrics, for characterizing object change processes. We use NOM (Number of Methods) for WMC, which means all weights are set to 1[1]. DIT is defined as the number of classes along the path from the class library to the target class, which represents readability[7]. Method lines of code (MLOC), class lines of code (CLOC), and the number of variables (NOV) are also used as the metrics.

Overview of a Case

The overview of the target system of the preliminary study is:

[*SystemA*]
- Number of versions: 4 versions
- Development term: 8 months

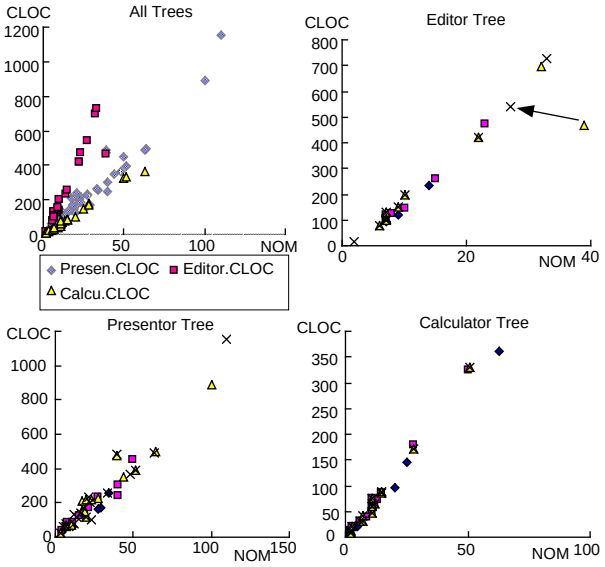

Figure 1: Scatter diagram of CLOC and NOM on SystemA

- Number of developers: 1
- Language: Visual Smalltalk
- Development process: incremental
- System function: editor for building simulation systems
- Size of the system: ver.4 has 52 classes, 927 methods and 8677 lines of code. They are 3.25 times, 5.56 times and 4.80 times as large as the corresponding value of the system of ver.1.

SystemA has been released to users once a month, and the engineer accepted new user requirements to be reflected on the system. Measurements are collected after the development of a prototype. There are following three class trees in *SystemA*:

- Editor Tree: a tree of classes for user interface to input initial data for simulation,
- Presenter Tree: a tree of classes for constructing simulation facilities visually and depict the result of the simulation, and
- Calculator Tree: a tree of classes for executing simulation algorithm.

Correlation analysis and tests are performed for these trees.

Correlation Analysis

Figure 1 shows scatter diagrams of CLOC against NOM for the three trees. The diagram at the top-left of Figure 1 shows points of all classes of the system for all versions. Other diagrams represent points of classes of each tree for all versions, respectively. The points in each diagram indicate that there is a high positive correlation between CLOC and NOM.

Table 1 shows correlation coefficients between CLOC and NOM for all classes and for each class tree. The correlation coefficient of each tree is positive and is higher than that of a set of all classes. We call the number of lines of code per method for a class $LOCpM_c$ and the mean of $LOCpM_c$ of classes in a tree $LOCpM_{tr}$.

Table 1: Correlation coefficients between CLOC and NOM per tree

tree	*all*	*Presenter*	*Editor*	*Calculator*
Correlation Coefficient	0.871	0.969	0.951	0.994

Table 2 gives common descriptive statistics of measurements of $LOCpM_c$ and shows that each tree keeps their $LOCpM_{tr}$ almost constant from ver.1 to ver.4.

We can conclude that a class tree should have its own peculiar value as $LOCpM_{tr}$. If there is an original value for every class tree and the value is stable as the system grows, there are possibilities that such knowledge would help engineers design classes.

Table 2: Distribution of $LOCpM_c$ for each tree defined in SystemA

Presenter Tree	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	7.63	8.44	8.63	8.93
Std Dev	1.89	1.27	1.62	1.78
Mini	5.55	6.10	6.00	6.00
Max	10.86	10.50	12.29	12.60
Data size	6	8	16	18
Editor Tree	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	15.10	16.34	16.32	16.24
Std Dev	2.16	2.63	3.11	4.07
Mini	13.14	13.14	12.00	7.00
Max	17.14	20.61	21.81	22.09
Data size	4	6	11	12
Calculator Tree	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	-	5.43	5.68	5.74
Std Dev	-	0.70	0.90	0.85
Mini	-	4.40	4.18	4.18
Max	-	6.33	7.00	7.00
Data size	-	6	15	15

Before evaluating the possibilities, we focus on a class change process. In Figure 1, an arrow in the editor tree diagram represents a direction of a class change from ver.3 to ver.4. The arrow indicates that $LOCpM_c$ of the class gets close to $LOCpM_{tr}$ of the tree. In the interview with the engineer, he said he had an impression that the class had been wrongly designed. Such impression has urged him to change the class. It implies that a hint of inappropriate design of a class could be obtained

objectively if the distance between the line and the point of a class were measured.

To examine if $LOCpM_{tr}$ is peculiar to a class tree, two-tail t-test was performed at 5% significance level for pairs of trees. Table 3 shows *p-values* calculated by the t-test. As a result, we can reject a null hypothesis: the values of $LOCpM_{tr}$ of class trees are the same, since all *p-values* are smaller than 0.05.

Table 3: *P-values* of two-tail t-test (5% significance level) for SystemA

metric	*Editor*	*Presenter*	*Calculator*
Editor	-	-	
Presenter	1.05E-15	-	
Calculator	2.22E-19	2.04E-16	-

To examine the stability of $LOCpM_c$, two-tail t-test and F-test are performed at 5% significance level for the mean and the variance of measurements of a version and the next version of each tree. Table 4 gives *p-values* calculated as the results of t-test and F-test. First, the table shows that there are not significant differences between the mean of measurements of a version and the next version of each tree, since *p-values* are greater than 0.05. Second, from the results of two-tail F-test, the table shows that there are not significant differences between the variance of measurements of a version and the next version of each tree. We cannot reject a null hypothesis: the mean and the variance are not changed by versions. Each tree has kept its variance during the system growth.

Table 4: *P-values* of two-tail t-test and F-test (5% significance level) for SystemA

t-test	*ver.1-2*	*ver.2-3*	*ver.3-4*
Presenter	0.229	0.470	0.662
Editor	0.459	0.989	0.963
Calculator		0.485	0.923
F-test	*ver.1-2*	*ver.2-3*	*ver.3-4*
Presenter	0.275	0.273	0.795
Editor	0.793	0.746	0.405
Calculator	-	0.590	0.771

Hypotheses of Class Tree Characteristics

To examine whether this kind of characteristic of a class tree can be found in general, we verify following three hypotheses:

1. A class inheritance tree has a peculiar value of $LOCpM_{tr}$, the mean of the number of lines of code per method,
2. The variance and the mean of $LOCpM_c$ of each class tree are stable during the system growth, and
3. The peculiar value does not depend on developers.

VERIFICATION OF HYPOTHESES

Overview of Cases

To verify the hypotheses, we study object change processes of the following systems:

[*SystemB*]
- Number of versions: 4 versions
- Development term: 8 months
- Number of developers: 1
- Language: Visual Smalltalk
- Development process: incremental
- System function: cash receipts transactions management system, its user interface is constructed by PARTS Workbench, a development environment of Visual Smalltalk.
- Size of the system: ver.4 has 62 classes, 2644 methods and 20470 lines of code. They are 1.68 times, 3.51 times and 3.07 times as large as the corresponding value of the system of ver.1,

[*SystemC*]
- Number of versions: 14 versions
- Development term: 3 months
- Number of developers: 4
- Language: Visual Smalltalk
- Development process: Waterfall
- System function: stock management systems.
- Size of the system: ver.14 has 133 classes, 1487 methods and 14934 lines of code. They are 1.51 times, 2.13 times and 2.27 times as large as the corresponding value of the system of ver.1.

SystemB was developed by an engineer incrementally. The system has been released to users once by two months, and the engineer accepted new user requirements to be reflected on the system.

SystemC was developed by four engineers and did not receive user requirements changes during the system growth. Class changes were derived from developers' design changes in the straightforward development process.

Case: SystemB

Overview of Class Trees We picked up the following four trees:

- TreeB1: a tree of classes that mediate between GUI objects and database objects,
- TreeB2: a tree of classes that manipulate persistent objects through putting them in a list,
- TreeB3: a tree of classes that represent persistent objects, and

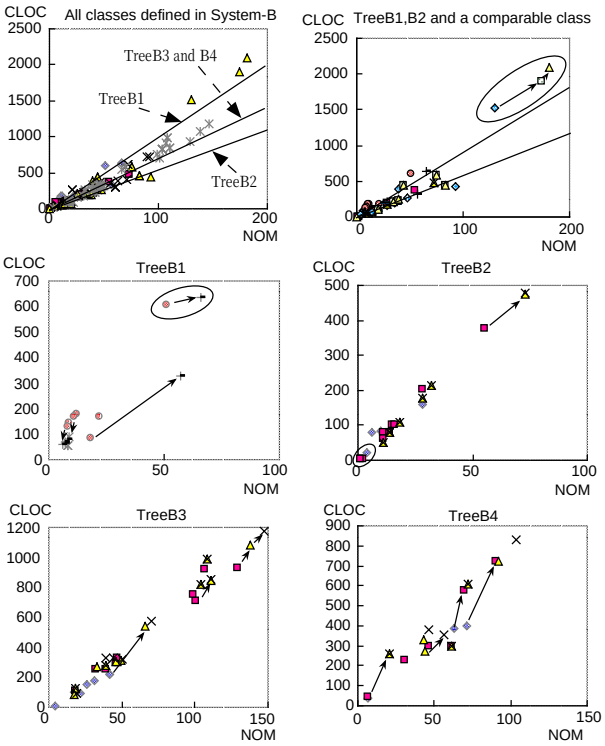

Figure 2: Scatter diagram of CLOC and NOM of SystemB

Table 5: Distribution of each tree defined in SystemB

TreeB1	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	8.04	12.29	9.06	9.07
Std Dev	1.80	4.53	1.64	1.64
Mini	7.00	4.56	5.74	5.74
Max	10.11	16.00	10.00	10.00
Data size	3	7	6	6
TreeB2	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	8.21	5.28	5.90	5.91
Std Dev	3.64	2.08	0.76	0.76
Mini	5.5	2.00	4.45	4.45
Max	13.33	7.27	6.66	6.66
Data size	4	11	7	7
TreeB3	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	4.97	7.08	7.37	7.49
Std Dev	1.26	0.78	1.10	1.13
Mini	2.25	6.17	5.12	4.95
Max	6.00	8.58	9.18	9.18
Data size	8	10	11	11
TreeB4	*ver.1.0*	*ver.2.0*	*ver.3.0*	*ver.4.0*
Mean	5.77	6.80	7.67	7.77
Std Dev	0.30	1.17	2.37	2.38
Mini	5.43	4.85	4.89	4.89
Max	6.10	8.26	12.29	12.29
Data size	4	7	7	7

- TreeB4: a tree of classes that construct tables by retrieving persistent objects.

TreeB1 and TreeB2 have a common superclass and TreeB3 and TreeB4 are trees expanding direct from the class Object.

Verification of Hypotheses

1. The first hypothesis

   Figure 2 shows scatter diagrams of CLOC versus NOM for a set of all classes and four trees of *SystemB*, respectively. A high positive correlation between CLOC and NOM can be observed in the diagrams. The results of correlation analysis show that the highest correlation coefficient is 0.994 of TreeB2, and the lowest is 0.915 of TreeB4. There is a high positive correlation between CLOC and NOM in *SystemB*.

   Table 5 gives common descriptive statistics of $LOCpM_c$. The results indicate that each class tree changes $LOCpM_{tr}$ from ver.1 to ver.2. Ignoring the data of ver.1 can be justified, because the size of the data of ver.1 is relatively small. In the case of TreeB2, since the tree was constructed in ver.2, four classes defined in the system in ver.1 have been redesigned to be subclasses of a newly added superclass in ver.2. Therefore, differences between the variance of ver.1 and ver.2 do not come from changes of TreeB2 characteristics but the redesigning processes.

   Figure 2 represents object evolution processes. An arrow represents directions of class changes. The gradients of the lines in the upper two diagrams indicate $LOCpM_{tr}$ of TreeB1 and TreeB2. The path shown in a circle in the upper right diagram is an evolution path of a class that shares the same superclass with TreeB1 and TreeB2 but is not a member of either tree. The distance of the path from the regression lines of TreeB1 and TreeB2 indicates the different characteristic of this class.

   In the scatter diagram of *TreeB1*, the middle left of Figure 2, circled points represent an evolution process of a class defined in ver.2. The tail of the arrow is not close to the mean of the TreeB1, but the head of the arrow indicates that the class has been changed to have a closer value to the tree peculiar value in ver.3.

   In the scatter diagram of each tree, the classes move their points along the line. Such class moves indicate that the tree preserves its peculiar value, $LOCpM_{tr}$, during the system growth.

   Two-tail t-test is performed at 5% significance level to study if there is a significant difference between $LOCpM_{tr}$ of pairs of trees. Table 6 gives *p-values* calculated by t-test for each pair of trees. There is not significant differences between $LOCpM_{tr}$ of TreeB3 and TreeB4, since the *p-value* is greater than 0.05. *P-*

values of other pairs of trees are smaller than 0.05, which are not regarded as the same mean. We can reject a null hypothesis: $LOCpM_{tr}$ is equal for class trees, since there are *p-values* smaller than 0.05. Therefore, the first hypothesis: a class inheritance tree has a peculiar value of the mean of the number of lines of code per method ($LOCpM_{tr}$), is verified.

Table 6: *P-values* of two-tail t-test (5% significance level) for SystemB

	TreeB1	*TreeB2*	*TreeB3*	*TreeB4*
TreeB1	-			
TreeB2	0.000	-		
TreeB3	0.000	0.004	-	
TreeB4	0.001	0.006	0.505	-

2. The second hypothesis

   Two-tail t-test and F-test are performed at 5% significance level to study if there are significant differences between the values of mean or variance of a version and the next version of each tree. The results are shown in Table 7. *P-values* calculated by t-test for each pair of trees are greater than 0.05, except the term from ver.1 to ver.2 of TreeB3. The exception can be ignored, because the size of the data of ver.1 is relatively small. These *p-values* indicate that there are not significant differences between the mean of a version and the next version.

   P-values derived from F-test for TreeB1 and TreeB2 in the term from ver.2 to ver.3 are smaller than 0.05, but other results shown in Table 7 are greater than 0.05. It means that TreeB1 and TreeB2 have changed their variance in the term from ver.2 to ver.3. Each Arrow in the diagram of TreeB1 in the middle left of Figure 2 represents a path of a class evolution from ver.2 to ver.3. If we imagine a line with the gradients of $LOCpM_{tr}$ in the diagram, the directions of the arrows seem to point to the line. When $LOCpM_c$ of classes of TreeB1 were not close to $LOCpM_{tr}$ of the tree in the early term, these classes have been redesigned in ver.3 and they got close to the tree peculiar value. $LOCpM_c$ of the circled two classes of TreeB2, defined in ver.2, are 2.00 and 2.50, respectively, and these classes were deleted in ver.3. These classes make the variance of TreeB2 smaller in the term from ver.2 to ver.3. If we ignore these two classes, *p-values* derived from F-test of TreeB2 become greater than 0.05 after ver.2. Now, we might accept a null hypothesis that the variance of a tree is stable between a version and the next version.

   The second hypothesis: the variance and the mean of $LOCpM_c$ of each class tree are stable during the system growth, is verified.

Table 7: *P-values* of two-tail t-test and F-test (5% significance level) for SystemB

t-test	*ver.1-2*	*ver.2-3*	*ver.3-4*
TreeB1	0.164	0.128	0.996
TreeB2	0.068	0.459	0.992
TreeB3	0.001	0.337	0.812
TreeB4	0.164	0.888	0.861
F-test	*ver.1-2*	*ver.2-3*	*ver.3-4*
TreeB1	0.284	0.041	0.998
TreeB2	0.157	0.022	0.991
TreeB3	0.196	0.394	0.926
TreeB4	0.144	0.151	0.983

Case: SystemC

Overview of Class Trees We focus on the following eight trees of *SystemC* developed by four engineers.

- TreesC1: a tree of stock domain object classes constructed by the following trees that inherit from a common superclass.
  - TreeC10: developed by Personα,
  - TreeC11: developed by Personα,
  - TreeC15: developed by Personα,
  - TreeC16: developed by Personβ, and
  - TreeC21: developed by Personβ.
- TreesC7: a tree of mediator object classes between persistent objects and the stock domain objects in memory, developed by Personγ,
  - TreeC70, and
  - TreeC71.

  These two trees also inherit from a common superclass.

  and
- TreeC82: a tree of objects that manage constant tables for tax rates, developed by Personδ.

Verification of Hypotheses

1. The first hypothesis

   Figure 3 shows scatter diagrams of CLOC versus NOM for a set of all classes and trees of *SystemC*. The high positive correlations between CLOC and NOM can be observed in the diagrams. The results of correlation analysis shows that the highest correlation coefficient of TreesC1 is 0.927, the lowest of TreeC82 is 0.561 and a correlation coefficient of all classes is 0.793. There is a positive correlation between CLOC and NOM in *SystemC*.

   In the bottom-left of Figure 3, a marked class lies in the position far from other points. Although it is a superclass of TreeC70 and TreeC71, the class cannot be considered as a class belonging to each of these trees.

   TreeC82 has lower correlation coefficient than other trees. Classes belonging to TreeC82 manage constant

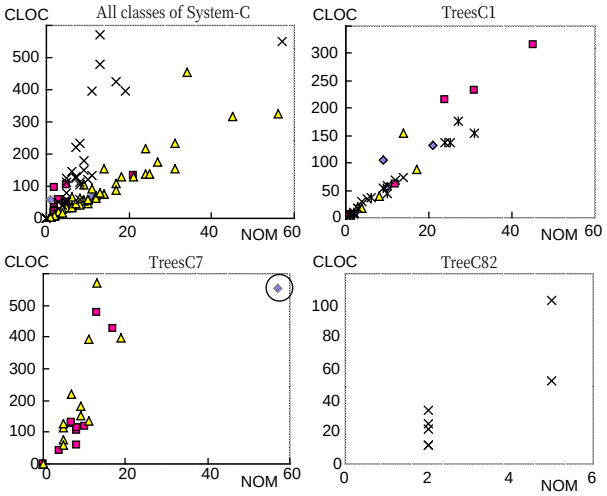

Figure 3: Scatter diagram of CLOC and NOM of SystemC

tables and provide methods to get values from the tables. These methods size depends on the table size, not on the responsibility of the class. TreeC82 is considered exceptional.

Table 8: Distribution of $LOCpM_c$ for each tree defined in SystemC

	TreeC10	*TreeC11*	*TreeC15*	*TreeC16*
Mean ($LOCpM_{tr}$)	4.87	5.17	5.99	5.58
Std Dev	0.98	1.23	2.54	1.98
Mini	4.00	4.00	4.00	4.00
Max	6.56	7.50	10.93	9.00
Data size	13	12	6	8
	TreeC21	*TreeC70*	*TreeC71*	*TreeC82*
Mean	4.90	17.30	23.28	16.44
Std Dev	2.20	8.97	10.10	13.70
Mini	4.00	7.38	11.60	6.00
Max	11.78	36.77	43.77	48.00
Data size	13	9	11	8

Table 8 gives common descriptive statistics of $LOCpM_c$ for all classes in the final version of *SystemC*. The results indicate that there are differences between $LOCpM_{tr}$ of class trees.

Two-tail t-test is performed at 5% significance level for the values of $LOCpM_{tr}$ of the trees. Table 9 gives *p-values* as the results of the test. The values show that there are two groups: one group is constructed by TreeC10, TreeC11, TreeC15, TreeC16 and TreeC21, and the other group is constructed by TreeC70 and TreeC71. *P-values* between trees belonging to the same group are greater than 0.05, but the values across the groups are smaller than 0.05. It means that there are significant differences between $LOCpM_{tr}$ of each group. Each group is constructed by the same class trees derived from the common superclass and is regarded as the big class tree.

Therefore, we can reject a null hypothesis: the means of lines of code per method are equal for class trees. The first hypothesis: a class inheritance tree has a peculiar value of the mean of the number of lines of code per method ($LOCpM_{tr}$), is verified.

2. The third hypothesis

   For the *SystemC*, we focused on differences between two engineers who developed classes of the same tree and studied if characteristic of engineers affect the value of the tree. The first engineer named Personα developed TreeC10, TreeC11 and TreeC15. The second engineer named Personβ developed TreeC16 and TreeC21. In Table 9, any differences between Personα and Personβ are not recognized. Therefore, we can accept a null hypothesis: engineers do not make differences to values of the same tree. The third hypothesis: the peculiar value is independent of developers, is verified.

DISCUSSIONS

We verified the hypotheses by two software development cases. From our studies, we found interesting values of class trees that are determined by the tree characteristics and are independent of developers. In the rest of this paper, we discuss the causes of the peculiar values and their effectiveness.

Causes of the Peculiar Values

Figure 4, Figure 5 and Figure 6 show histograms of frequency of MLOC, method lines of code, with cumulative frequency for each tree of each system, respectively. Every diagram has the very similar shape that has a peek on the left edge and a long tail on the right.

The differences between the values of trees do not come from the shape of the histogram nor the position of the peek, but from the length of the right side tail. Calculator tree of *SystemA* has the smallest mean in the studied trees and TreesC7 of *SystemC* has the greatest. Differences between the figures of two trees are obvious. Frequencies of MLOC in Calculator tree gathered at smaller numbers than those in TreesC7.

If a method has a complex role (i.e., manipulate various objects and answer the calculated results, manages objects relationships), MLOC must be great. If a class has a complex role, the number of lines of code per method must be great. Such characteristic of a class is common to the classes in the common tree, because the inheritance tree is constructed by classes with a similar behavior. Consequently, $LOCpM_{tr}$ is determined as the characteristic of a tree. The characteristics cannot

Table 9: *P-values* of two-tail t-test for SystemC

Tree	*C10*	*C11*	*C15*	*C16*	*C21*	*C70*	*C71*	*C82*
C10								
C11	0.510							
C15	0.337	0.483						
C16	0.369	0.610	0.752					
C21	0.964	0.708	0.389	0.474				
C70	0.003	0.004	0.005	0.004	0.003			
C71	0.000	0.000	0.000	0.000	0.000	0.178		
C82	0.049	0.053	0.068	0.062	0.050	0.882	0.254	

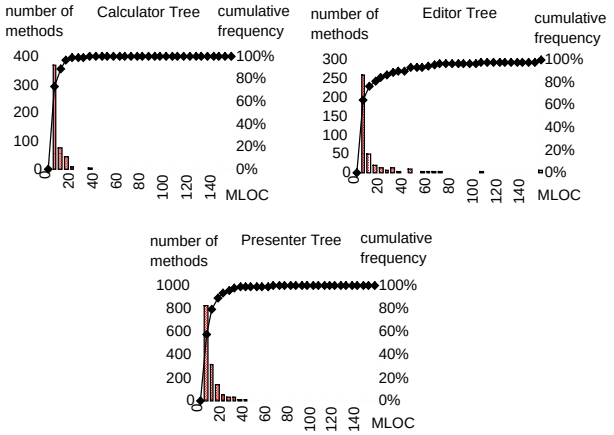

Figure 4: Histogram of MLOC of SystemA

be changed unless the behavior of classes of the tree changes.

The results of our study can fit in with the developers' technical class categorizations.

Effectiveness of Observations for the Developers
From our studies, the first hypothesis: a class inheritance tree has a peculiar value of lines of code per method, is verified. If a class is added to a class tree, the class must have the value close to $LOCpM_{tr}$ of the tree sooner or later. If $LOCpM_c$ of the class is not close to the peculiar value of the tree, the distance between these two values may imply poor design. Our future work is to verify the truth of the law: the characteristic of a class inheritance tree help engineers evaluate their class design properness quantitatively.

Evolution of Class Trees and Classes
M. M. Lehman and L. A. Belady studied system evolution and discovered laws of software evolution[2, 5]. They proposed laws of "continuing change" and "increasing complexity". Here, we focus on the evolution processes of classes and their inheritance trees and try to clarify the characteristics to be changed and not to be changed. Object-oriented systems tend to be devel-

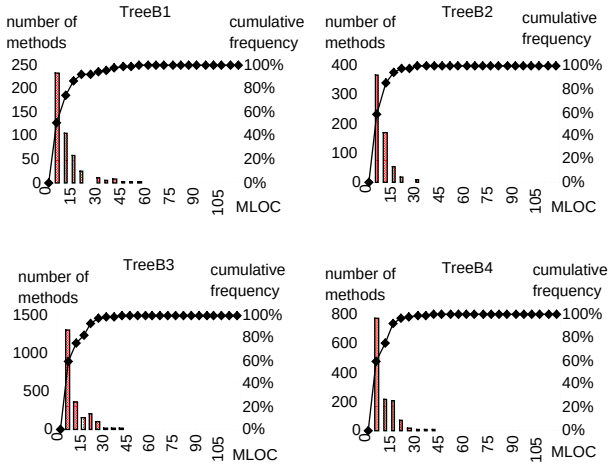

Figure 5: Histogram of MLOC of SystemB

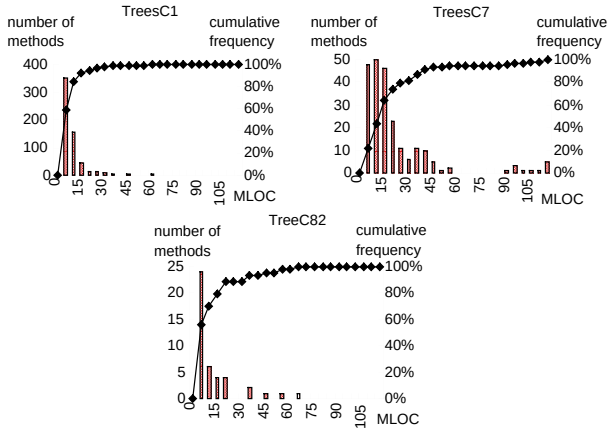

Figure 6: Histogram of MLOC of SystemC

oped incrementally[4] and is often redesigned to be more robust systems for user requirements changes. Along the system growth, some classes change their complexity continuously with keeping distribution shape of measurements of the whole systems [7]. This characteristic follows the law of software evolution.

We want to discuss the law of inheritance tree evolution that exists only in object-oriented software. From our observations, the trees can keep their peculiar values and their distribution along the system growth and the classes in the tree change their values towards the direction of the lines determined by the tree peculiar values. The characteristic of class inheritance trees are considered as class evolution circumstances which affect the ways of their evolutions.

Class tree evolution processes need to be studied in detail qualitatively to verify the second hypothesis. Trees may have their own evolution processes, though we focus on the general evolutional characteristic in this paper.

CONCLUSIONS

We verify hypotheses by studies of three systems. From the results of our studies, the peculiar values of class inheritance trees is considered to be stable through the system growth. Our next studies will be check the validity of the effectiveness of the peculiar value of class trees on evolutional design processes.

Acknowledgment

This work is supported by the Advanced Information Technology Program (AITP) of Information-technology Promotion Agency (IPA), Japan. We also thank to Mr. Atsushi Tomoeda, Ms. Harumi Matsuda and Mr. Hirotsugu Kondoh of SRA Co.,Ltd. for their help to collecting measures.

REFERENCES

1. Basili, V. R. and Melo, W. L. A Validation of Object-Oriented Design Metrics as Quality Indicators, *IEEE Transactions on Software Engineering*, 22,10 (1996), 751–761.
2. Lehman, M. M. and Belady, L. A. *Program Evolution*, Academic Press (1985).
3. Chidamber, S. R. and Kemerer, C. F. A Metrics Suite for Object Oriented Design, *IEEE Transactions on Software Engineering*, 20, 6 (1994), 476–493.
4. Henderson-Selleres, B. and Edwards, J. M. The Object-Oriented System Life Cycle, *Communications of ACM*, 33, 9 (1990), 142–159.
5. Lehman, M. M. Programs, Life Cycles and Laws of Software Evolution, in Lehman, M. M. and Belady, L. A. ed. *Program Evolution*, Academic Press (1985), 393–449.
6. Lorenz, M. and Kidd, J. *Object-Oriented Software Metrics*, Prentice Hall (1994).
7. Nakatani, T., Tamai, T., Tomoeda, A. and Sakoh, H. Quantitative Analysis on Evolution Process of Object-Oriented Systems, *Proc. of the International Symposium on Future Software Technology* (1996), 49–56.
8. Nakatani,T., Tamai, T., Tomoeda, A. and Matsuda, H. Towards Constructing an Object Evolution Model *Proc. of Asia-Pacific Software Engineering Conference '97*, IEEE (1997). (in Printing)