Quantitative Analysis on Evolution Process of Object-Oriented Systems

Takako NAKATANI[†], Tetsuo TAMAI[‡],
Graduate School of Arts and Sciences
University of Tokyo[§]

Atsushi TOMOEDA[¶] and Hiroshi SAKOH[‖]
SI Business Department,
SRA Co., Ltd.[**]

Abstract

In object-oriented systems development, we can build a prototype in a short period of time by reusing library classes or components. Prototyping approach has obtained the variety from the throwaway type to the evolutionary type. When we develop systems incrementally by ensuring users' needs, we must consider reconstruction of the systems depending on their extensibility. Though we develop a system incrementally, it is not evident when we must scrap the system and migrate to a new system. In not a few cases, problems occur when a small-scale system grows to a large-scale system.

To understand system evolution processes we define evolution metrics at four levels, i.e. the whole system level, the class level, the message level and the method level. Then we measure real systems by applying the evolution metrics and quantitatively analyze the relation between measures and evolution processes. As results, we can show effectiveness of the evolution metrics to represent system evolution process quantitatively.

1 Introduction

Software systems are developed to satisfy users' needs, under a set of constraints such as functionality, reliability, time to deliver, cost, and prospected life time. The incremental development approach has been considered to have such an advantage that the system can correspond to users' needs quickly and continuously[3]. In 1990, Tate said "evolutionary prototyping, in the sense that the prototype evolves into the production system should, and perhaps will in future, simply be called evolutionary development.[13]" As the incremental development approach has been applied to more system development projects, the differences between the development process and the maintenance process becomes much smaller. Thus, a framework for understanding long-term system changing dynamics is required and the notion of system evolution process may fit to that purpose.

The incremental development approach is quite natural with object-oriented system development. Though object-oriented development process includes processes to redesign classes to develop reusable library classes or components for the future system extension or new development[1][4], it is hard to distinguish between the development process and the redesigning process. When we observe object-oriented system evolution process, we observe the development process, the maintenance process, and the redesigning process.

The evolution process contains many kinds of activities, but it is possible to identify the characteristics of the activities by measuring their products. We measure and analyze evolution processes, aiming at the following objectives:

- representing the evolution process quantitatively,
- extracting software components, like classes and/or methods, that had better be paid particular attention for future system evolution.

By achieving the second objective, it will be possible to help developers decide which step to take, redesigning step, migration step or the next incremental step forward when the system meets new specifications. To achieve these objectives, we define a set of metrics, called *evolution metrics*, to measure system evolution process quantitatively by applying them to a real application system. In this paper we define "*metric*" as a scale to characterize attributes, and "*measure*" as a numerical value for an attribute assessed for magnitude against an agreed scale[5].

[†]tina@graco.c.u-tokyo.ac.jp
[‡]tamai@komaba.ecc.u-tokyo.ac.jp
[§]3-8-1 Komaba, Meguro-ku, Tokyo 153, Japan
[¶]tomoeda@sra.co.jp
[‖]sakoh@sra.co.jp
[**]3-12 Yotsuya, Shinjuku-ku, Tokyo 160, Japan

This paper is organized as follows. In section 2, we describe evolution metrics. In section 3, we show the results of experiments of applying the evolution metrics to a real system and analyze the system evolution process inferred from the obtained measures. In section 4, we discuss related works. Finally in section 5, we draw some conclusions from our work.

2 Evolution Metrics

To measure object-oriented system evolution process, we define four observation levels: the system level, the class level, the message level, and the method level, and determine evolution metrics at each level as follows.

1. The system level

   NCD: the number of classes defined by engineers; SLOC: total lines of code summing up lines of code of all classes; SNOM: the number of methods defined in the system; NMN: the number of message names defined in the system; NCT: the number of class trees consisted of application classes which have superclasses in the common library; and NAC: the number of abstract classes which have subclasses.

2. The class level

   CLOC: lines of code to define methods of each class; NIV: the number of instance variables; CNOM: the number of method definitions; DIT: the depth of inheritance tree (LDIT: the depth from the top of the total class hierarchy; and ADIT: the depth from the lowest library class); NOD: the number of descendant classes.

3. The message level

   NRM: the number of classes which can respond to the message; NOM: the number of method definitions corresponding to the message name.

4. The method level

   MLOC: source lines of code to define each method.

With these evolution metrics, properties of the whole system and its components become measurable.

These evolution metrics lead to inclusion relations between the levels. For a system with classes C_1, C_2, ..., C_n, and for a class C_i with methods Mi_1, Mi_2, ..., Mi_j, consider the number of lines of code of class C_i, $CLOC_i$ and the number of lines of code of the system, $SLOC$. $SLOC$ and $CLOC_i$ are related as:

$$SLOC = \sum_{i=1}^{n} CLOC_i.$$

If we denote the number of lines of code of the j th method of class C_i by $MLOCi_j$ then

$$CLOC_i = \sum_{j=1}^{n_i} MLOCi_j,$$

where n_i is the number of methods of the class C_i.

A similar relation also holds for the number of methods. For a class C_i with a set of methods MD_i, the total set of methods MD is,

$$MD = \bigcup_{i=1}^{n} MD_i.$$

Similarly, for a set of message names defined in a class C_i, MS_i, the total set of messages MS is,

$$MS = \bigcup_{i=1}^{n} MS_i.$$

In general,

$$|MS_i| = |MD_i|,$$

but

$$|MS| \leq |MD|,$$

because MS is not necessarily a direct sum of MS_i, while MD is always a direct sum of MD_i. The case $|MS|$ is equal to $|MD|$ occurs only when there is no polymorphism in the system.

Other class evolution metrics, NIV, LDIT or ADIT, and NOD, are measured only at the class level. We can observe the statistic distribution of these metrics over the classes at the system level.

3 Experiment

3.1 Overview of the Measured System

The evolution metrics are applied to a thermo-control simulation system built on PARTS Workbench, which is a part of VisualSmalltalk. The system was developed by one engineer and took eight months from the analysis to the final delivery. During this period, the system was delivered six times to the user. Whenever the system was delivered, the user evaluated it to decide new requirements for the next step. Since this system was developed step by step to satisfy user's needs, its development process can be viewed as an evolution process.

The final structure of the system consists of (see Figure 1):

Figure 1: Overview of the system

- presentation parts: the parts to construct simulation facilities visually and depict the result of the simulation,
- editing parts: the parts to input initial data for simulation, and
- calculation parts: the parts to execute the thermo-control algorithm.

The first two kinds of parts were built on PARTS Workbench. The final system ver.5 has 52 classes (see Table 1. Ver.4 has the same number of classes). It took two months from the delivery of the system ver.0 to that of ver.1 and one month each for the delivery of the following versions.

Overview of each version is as follows.

- ver.0
  A prototype was developed to explore the possibility that VisualSmalltalk could be applied to the simulation system. It defined some parts for simulation using PARTS Workbench, and showed its capability of connecting these parts dynamically through graphical user interface and displayed the results of a simulation according to their properties.
- ver.1
  A minimum set of classes to execute the simulation is constructed from presentation parts and editing parts.
- ver.2
  The more diverse the components that construct simulation became, the more complex algorithm was being required. Thus, a set of calculation parts were newly defined by decomposing the set of presentation parts to implement the simulation algorithm. Thus, the basic structure of the system as shown in Figure 1 was established.

Figure 2: Evolution process of the whole system

- ver.3
  As simulation components became more diverse and thermo-controlling mechanism was required to be more complex, the graphical user interface was changed to make the operation of the system not too complex.
- ver.4
  Some components were reorganized.
- ver.5
  The system usability and reliability were improved.

The system ver.0 was a prototype and its system structure was different from the others. Ver.5 has almost the same structure as ver.4 and little difference is observed between them. Therefore, we select four versions for our evolution study from ver.1 to ver.4.

3.2 The Result and Analysis

3.2.1 System Level Evolution Process

1. Analysis of the evolution process

   We try to relate the evolution metrics, NCD, SLOC, SNOM, NMN, NCT and NAC, with system evolution. Evolution stages are measured from version to version by these metrics and S-shaped curves are plotted as shown in Figure 2 (see also Table 1). It can be interpreted that requirements grew during the period from ver.2 to ver.3 and after ver.3, requirements growth came to the relatively stable stage. SLOC, the total lines of code, shows sensitive reaction to the requirements changes, while NCT, the number of class trees, shows less reaction.

Table 1: The system level evolution process

metric	*ver.1*	*ver.2*	*ver.3*	*ver.4*
NCD	16	26	47	52
SLOC	1560	3714	8044	8677
SNOM	193	436	885	927
NMN	115	241	452	463
NCT	5	6	6	6
NAC	4	5	9	11

Table 2: The number of classes contained in trees

Tree name	*ver.1*	*ver.2*	*ver.3*	*ver.4*
Editing-1	1	1	1	1
Editing-2	3	5	10	11
Presen-1	1	1	1	1
Presen-2	6	8	16	18
Calculation	–	6	14	16
Phys-Const	5	5	5	5
total	16	26	47	52

Table 2 shows the numbers of classes belonging to class trees defined in each version. In ver.2, a class tree named Calculation appears as a new family defined by decomposing Presen-2 of ver.1. This kind of design decision process may also be noticed by observing the change in the number of class trees, NCT in Table 1. After ver.2, the engineer did not add any class trees but added new classes as subclasses of the existing classes.

The evolution processes of the class tree structure are schematically depicted in Figure 3. In this diagram, a dark gray circle represents a newly added class and a light gray circle represents a redesigned class which has changed its hierarchical position in the class tree or has changed its structure. It can be seen that a newly added class is defined as a subclass of an existing application class, and a new class tree is not added except *C-1* in ver.2. Furthermore, a class is sometimes redesigned when its subclass is added. These observations may imply that the design process of the simulation system consists of:

(a) defining application classes under library classes (see Figure 3 phase 1);

(b) defining a new class tree (see Figure 3 light gray circles in phase 2);

(c) adding a new class under an existing application class (see Figure 3 dark gray circles);

(d) redesigning an existing class, adding a new class as its subclass if necessary (see Figure 3 light gray circles in phase 3).

Figure 3: Evolution process of class trees

Only the first type of activities in this design process can be inferred from the increase in the numbers of classes contained in trees in Table 2.

Table 3: Statistics of MLOC collected from ver.1 to ver.4

	ver.1	*ver.2*	*ver.3*	*ver.4*
Mean	8.1	8.5	9.1	9.4
Median	5	5	5	5
Std. dev.	10.8	16.0	19.5	21.5
85%border	12	11	12	12
Minimum	3	3	3	3
Maximum	116	261	410	427

As an evolution sample, we draw histogram and cumulative distribution of MLOC (see Figure 4). Table 3 shows the statistics of MLOC collected from ver.1 to ver.4. In Table 3 and Figure 4, each version shows that the

Figure 4: Histogram and cumulative distribution of MLOC

number of MLOC on the 85% border is equal to 11 or 12, the minimum is 3, the median is 5, and the mean is between 8 and 10. It means the measure of MLOC has relatively stable distribution shape during these periods, but a few methods with exceptionally large size are observed and the maximum measure is continuously growing in Figure 4. The existence of these exceptional values may indicate some anomaly in the system design[8]. In the conventional structured programming, a guideline of keeping the maximum number of source lines of code per module to say 50 is often recommended, because it fits in a standard-sized paper and can be read at a glance. Following this conventional coding style, the desirable value of MLOC for Smalltalk may be less than 10, because it can be read at a glance on a *System Browser* window.

These exceptionally large methods are observed only in few classes and they are growing even larger as the system evolves (see Figure 5). This may be an outcome of a design policy to enhance the system and satisfy the users' needs as quickly as possible.

By interviewing with the engineer, we found that the classes with huge sized methods were identical to those that had been listed up by the engineer himself as candidates to be redesigned. It means the evolution metrics can capture a part of the system to be redesigned.

2. Effectiveness of the evolution metrics

The evolution metrics have some advantages in letting us understand the system evolution process through:

- clarifying the evolution volume over a period of time;
- identifying the existence of methods and classes which have measures relatively close to the mean values and are stable throughout the system evolution;
- identifying the existence of methods and classes which have a tendency of growing along with the system evolution; and
- alarming the existence of methods or classes which may be required to be redesigned.

3.2.2 The Class Level Evolution Process

1. Analysis of the evolution process

   Lines in Figure 5 plot the measures of CNOM from ver.1 to ver.4. They show that there are two groups of classes; one is a group of stable classes and the other is a group of classes with fluctuating values. Since CNOM represents the size of responsibility of a class, measures of CNOM can be regarded as representing the volume of users' requirements and/or designer's requirements.

   The following three classes are typical of the unstable group, which have been detected by their peculiar changes in their measures over time.

   - *Class1*: a calculation class for thermo-exchange

     It plays a central role in the thermo-control simulation system and appears from ver.2 after the first redesigning process of dividing Presen-2 into two classes. Its responsibilities are decreasing during the period from ver.2 to ver.3. Indeed, when we observe CNOM belonging to each class tree, this class has the biggest measure 63 over 2.5 times bigger than the next biggest class measure 25 in the Calculation tree. In ver.3, the engineer developed a new class as its superclass by generalizing two classes including *Class1*. Therefore, the measure of CNOM of *Class1* decreased. This redesigning process is shown in Figure 3 from ver.2 to ver.3 and redesigned classes are represented by double circles.

Table 4: Measures of correlation coefficient between class evolution metrics(ver.3)

metric	*CLOC*	*NIV*	*CNOM*	*ADIT*
CLOC	1			
NIV	0.59	1		
CNOM	0.89	0.84	1	
ADIT	-0.18	0.27	0.04	1

Figure 5: Class evolution process (CNOM)

- *Class2*: a presentation class for *Class1*
  It is defined to manipulate *Class1* for users through graphical user interface. The increase of the specification volume from ver.2 to ver.3 may have influences on *Class*2 directly. It was not redesigned during the period from ver.2 to ver.3, but *Class1* was.
- *Class3*: a presentation class for a pipe which connects simulation parts
  It appears in the tree Presen-2 at ver.1, implying it is produced by the first redesigning process of dividing Presen-2 into two trees. It is conceivable that the redesigning processes decrease the measure of *Class3* from ver.1 to ver.2.

From their evolution process, the classes classified into the unstable group tend to increase their responsibility unless their class family structure is redesigned.

Our measurement detects *Class1*, *Class2* and *Class3* as classes with relatively large measures of CNOM, NIV and CLOC. The measures of CNOM, NIV and CLOC have high correlation coefficients between them. We show the correlation coefficients of CNOM, NIV, CLOC and ADIT in Table 4. It shows low correlations of ADIT with the other metrics. We selected the pair of CLOC and CNOM that have the highest correlation coefficient and plotted points corresponding to classes(see Figure 6).

Figure 6: Scatter plot of CLOC and CNOM

We can see a clear linear relation between these two values for classes of each class tree. Table 5 shows regression coefficients between CLOC and CNOM, which correspond to the gradients of these lines, indicating the mean values of MLOC for each class tree.

It is interesting that in the scatter plot we can find one point obviously far away from the line in the Editing parts tree. The class corresponding to this point was first defined in ver.3 and was redesigned in ver.4, to be plotted back on the line.

By drawing the scatter plots belonging to class trees, we have detected a class which had peculiar design characteristics and observed that the class had been redesigned to have normal measures.

This observation suggests that there are some

Table 5: Regression coefficients between CLOC and CNOM

Tree name	*ver.1*	*ver.2*	*ver.3*	*ver.4*
Presen − 2	6.80	7.93	8.69	9.31
Editing-2		18.39	19.88	20.07
Calculation		5.67	6.23	6.20

implicit design rules for each class family that govern the properties of its classes or methods. Finding such rules may lead to the discovery of exceptional and maybe erroneous design parts.

2. Effectiveness of the evolution metrics

   The evolution metrics have some advantages in letting us understand the class evolution process through:

   - clarifying the class evolution volume over a period of time;
   - identifying classes which have measures relatively close to the mean values and are stable throughout the system evolution;
   - identifying classes which have a tendency of growing along with the system evolution; and
   - alarming classes which may be required to be redesigned and indicating required redesigned direction.

In the same way, we can show the effectiveness of the method level evolution metrics to identify methods which have measures relatively close to the mean values and are stable throughout the system evolution or to identify methods which have a tendency of growing along with the system evolution.

4 Related Work

Henderson-Sellers, B. and Edwards, J. M. compared object-oriented systems life cycle with traditional software life cycle[4]. They depicted object-oriented life cycle by the fountain model and showed interaction processes between requirements evolution and design and implementation.

Many researches of object-oriented metrics have been conducted from early 1990's[14]. Chidamber, S. R. and Kemerer, C. F. defined six metrics, called CK metrics, to determine the quantity of class coupling, class cohesion, method complexity, and understandability of classes for reuse[2].

Lorenz, M. and Kidd, J. discussed many metrics for managing projects and estimation[8]. Our results show almost the same mean values of class level metrics like CLOC, CNOM or NIV for every version as those given in their book.

Since our target system was developed in Smalltalk, it was easy to measure the metrics by using Metalevel facilities[6]. However, since Smalltalk has the complete dynamic binding mechanism, the cohesion and coupling of objects[10] can be measured only at the run-time system. We are now trying to analyze dynamic metrics of a system at run-time.

Stark, G. and his colleagues applied metrics for the systems management. They said "managers do not need exact models or metrics to make decision and excellent and useful results are regularly obtained from simple models."[11] Furthermore, they discussed effectiveness of measurement over time.

Lehman defined five laws for the system evolution[7]. His second law was "increasing complexity: the number of superimposed changes increases, the system becomes more complex." According to his law, three classes detected from measures of CNOM over time might make the system complex. But in object-oriented systems, since the redesigning process is usually involved, the system does not become complex straightforwardly. By redesigning the system, its life time gets longer, but it sometimes requires more cost.

Tamai, T. and Torimitsu, Y. conducted a survey of software lifetime[12]. As the results of their survey, they noted that software replacement could bring benefits by introducing new technologies and educating inexperienced engineers. When we use object-orientation, we can add other benefits from rebuilding an application framework that makes it possible to get more robust systems and to satisfy users' requirements more quickly. We can see this kind of evolution in the redesigning process from ver.1 to ver.2 of our case study, but the redesigned volume is too small to call it replacement.

5 Conclusion

This paper described the results of experiments and quantitative analysis of system evolution processes over a period of time. Software development usually has some specific objectives, i.e. to examine its feasibility and usability to satisfy users' needs keeping extensibility. We think there is a different evolution process for each software development achieving these objectives. To understand the system evolution process, we defined and applied evolution metrics to the real system. After analyzing the values measured by evolution met-

rics, it became clear that the effects of measuring system evolution process over time were clarified:

- by measuring system evolution in terms of SLOC, SNOM, NMN, NCT and NAC, we can infer developed volume;
- by measuring class evolution in terms of CLOC, NIV, CNOM, and ADIT, we can classify classes into an unstable group and a stable gtoup;
- by quantitatively analyzing measures, we can extract some software components, classes and/or methods that have evolved their size continuously and have large size, and others that have changed little and have kept their size rather small;
- by researching class evolution process, we can identify characteristic values like CLOC per CNOM for each class hierarchy;
- by identifying specific values for each class hierarchy, we can point out irregular classes and suggest class hierarchy dependent values as implicit design norms.

Thus, we have showed that the evolution metrics can be used to represent the evolution process quantitatively and to extract software components, like classes and/or methods, that had better be paid particular attention for further system evolution. We must observe more cases to deepen our understanding of object evolution processes.

Acknowledgment

This work is supported by the Advanced Information Technology Program (AITP) of Information-technology Promotion Agency (IPA), Japan.

References

[1] Bollinger, T. B. & Pfleeger, S. L. : "Economics of Reuse: Issues and Alternatives," *Information and Software Technology*, Vol. 32, No.10 (1990), pp. 643-652.

[2] Chidamber, S. R. & Kemerer, C. F. : "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol.20, No.6 (1994), pp. 476-493.

[3] Davis, A. M. : "A Strategy for Comparing Alternative Software Development Life Cycle Models," *IEEE Transaction on Software Engineering*, Vol. 14, No. 10 (1988), pp. 1453-1461.

[4] Henderson-Selleres, B. & Edwards, J. M. : "The Object-Oriented System Life Cycle," *Communications of ACM*, Vol. 33, No. 9 (1990), pp. 142-159.

[5] Henderson-Sellers, B. : *Object-Oriented Metrics –Measures of Complexity–*, Prentice Hall, 1996.

[6] LaLonde, W. & Pugh, J. : "Gathering Metric Information Using Metalevel Facilities," *Journal of Object-Oriented Programming*, March-April, 1994, pp. 33-37.

[7] Lehman, M. M. : "Programs, Life Cycles and Laws of Software Evolution," in Lehman, M. M. & Belady, L. A. ed. *Program Evolution*, Academic Press, 1985, pp. 393-449. (It is reprinted from Proc. IEEE Spec. Issue on Software Engineering, Vol. 68, No. 9 (1980), pp. 1060-1076.)

[8] Lorenz, M. & Kidd, J. : *Journal of Object-Oriented Software Metrics*, Prentice Hall, 1994.

[9] Nakatani, T., Tomoeda, A., Sakoh, H. & Tamai, T. : "Examination of Evolution Metrics for Object-Oriented Systems," *Proc. of Software Symposium '96*, SEA, 1996, pp. 52-62 (In Japanese).

[10] Schach, S. R. : "The Cohesion and Coupling of Objects," *Journal of Object-Oriented Programming*, January, 1996, pp. 48-50.

[11] Stark, G., Durst, R. C. & Vowell, C. W. : "Using Metrics in Management Decision Making," *IEEE Computer*, September, 1994, pp. 42-48.

[12] Tamai,T. & Torimitsu,Y. : "Software Lifetime and its Evolution Process over Generations," *Proc. of the Conference on Software Maintenance*, November, 1992, pp. 63-69.

[13] Tate, G. : "Prototyping: Helping to Build the Right Software," *Information and Software Technology*, Vol. 32, No. 4 (1990), pp. 237-244.

[14] Whitty,R. : "Object-Oriented Metrics: A Status Report," *Object Expert*, Jan./Feb., 1996, pp. 35-40.