# Metrics and Analysis of Software Architecture Evolution with Discontinuity

**Mikio Aoyama**
**Department of Information and Telecommunication Engineering**
**Nanzan University**
**27 Seirei, Seto, 489-0863, Japan**
**mikio.aoyama@nifty.com**

## Abstract

*This article proposes a set of metrics for software architecture evolution and discusses continuous and discontinuous software evolution with the metrics proposed. We claim that discontinuity arises to reengineer software architecture and is an essential aspect of software evolution. We expect the proposed metrics can help to identify the architecture preservation core to be preserved over certain period of evolution, and the active evolution zone frequently changing during evolution. The evolution dynamics with discontinuity reveals the non-homogeneous nature of software evolution over the space and time.*

## Keywords

Software evolution, software architecture, software pattern, software metrics.

## 1. Introduction

This article proposes a set of metrics for software architecture evolution and discusses discontinuity of software evolution with the metrics proposed.

Software evolution is collective changes made on to software systems in order to meet the change of requirements and/or surrounding environments.

Evolution is an essential nature of software systems since living systems have to adapt to changing world. In today's rapid change of technical and business climates, which software systems have to live with, software evolution is one of central concerns we have to pay attentions.

From our observation of the evolution of several software systems over years, we found two patterns of evolution of *continuous* and *discontinuous* as illustrated in Figure 1 [3]. And, there is a pattern of emergence of two evolution patterns. Within our observation records, we found discontinuous evolution emerges between certain periods of successive continuous evolution as if the discontinuous evolution interrupts continuous evolution. Not only our evolution records, this pattern can be found in several software evolution histories including early evolution of OS/360 [8], evolution of multiple business applications [7], feature evolution of telephone switching software [1], and evolution of Linux [5].

In the study of genetic evolution, Eldredge and Gould proposed *Punctuated Equilibrium Theory* to model a phenomenon of discontinuity [4]. Punctuated Equilibrium Theory claims that a gene changes its essential aspects to adopt the environmental change and emerges as a new gene. This triggers discontinuous, i.e. punctuated, evolution in order to survive the environmental change and reach another equilibrium status. Although the continuity and discontinuity are still under debate in genetic evolution research community, we observe many corresponding aspects between genetic evolution and software evolution.

The subject of this research is to provide a model of discontinuity of software evolution based on Punctuated Equilibrium Theory. Like genetic evolution, we found that the discontinuous evolution is architectural [3]. So, we explore metrics and mechanism of software architecture evolution.

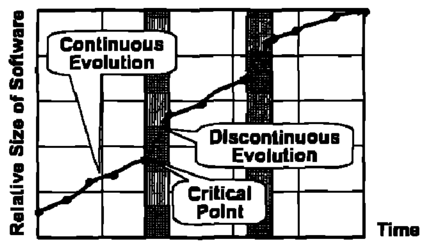

**Figure 1 Continuous and Discontinuous Evolution**

## 2. Metrics of Software Architecture Evolution

We propose a set of metrics for evaluating software architecture evolution based on the structural similarity metrics.

For evaluating similarities of two structured objects, such as trees and graphs, there are number of metrics and algorithms proposed [9, 10]. To derive metrics for software architecture evolutions from these metrics, we employ tree as a simplified model of software architectures since tree model can be abstraction of class diagrams or tree-structure charts.

We assume evolutional change of a software system as a set of changes made on the tree. Thus, for the metrics of evolutional change of a software system, we propose a similarity measure of two trees in terms of cost of change operations made on the trees.

### 2.1 Tree Model of Software Architecture

Figure 2 illustrates a model of software evolution, which is commonly used. We assume *an* evolution is a map between two systems [6]. Here, a system is modeled as a tree. The node and edge respectively represents component and connector. The evolution is a set of collective changes made on nodes and edges by adding, removing, or modifying them, while some parts of the system are unchanged [2].

To evaluate the process of software evolution, it's commonly used to measure the change by the amount of source codes changed. However, the evolution may arise at the different levels of abstraction, such as, at architecture, or at classes and modules. Since evolution is not simply a change of a module, but a set of collective changes made on modules, it's important to evaluate the structural properties of change over the evolving period. Thus, we propose a set of metrics for software architecture evolution based on the similarity of tree structures.

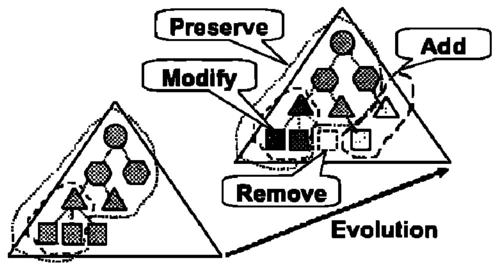

**Figure 2 A Simple Model of Software Evolution**

### 2.2 Evolution Metrics based on the Structural Similarity of Software Architectures

A number of researches have been done on the distance, i.e. similarity, metrics of structures, such as graphs, trees, and sequences [10]. Since we adopted tree as a model of software architecture, we introduce a set of distance metrics for trees as the base metrics of software architecture evolution.

There are two categories of distance metrics for trees. One category is based on the direct mapping between trees [9]. The other one is based on the cost of rewriting operations on the tree.

(1) Evolution Cost Metrics

For software architectures, evolution is made by a set of change operations on the components and connectors. As illustrated in Figure 2, we assume software architecture as a tree comprising of a set of components and connectors connecting them. Each component and connector is identified with a unique identifier, label.

To measure the evolution of software architectures, we introduce the metrics based on the rewriting operations on the trees. As commonly used, we employ following three types of operations on the components and connectors.

1) Add: Adding new components or connectors.

2) Remove: Remove components or connectors.

3) Modify: Modify the components or connectors.

Thus the cost of a set of changes is defined by equation (1).

$$C_0 = W_A * \sum_i A_i + W_R * \sum_j R_j + W_M * \sum_k M_k \qquad (1)$$

Where, $A_i$, $R_j$ and $M_k$ respectively denotes the cost of Add, Remove, and Modify operation, and $W_A$, $W_R$, and $W_M$ respectively denotes the weight of the cost.

(2) Metrics of Architecture Preservation over the Evolution

Now, we introduce a metric of architecture preservation factor, APF in short.

Assume all the elements in a tree are preordered. And, assume there is a mapping from an element $E_i$ in the tree $T_N$ to an element $E_j$ in the tree $T_M$. Denote such a mapping as a pair (i, j). So, we denote the collection of all such mappings (i, j) as $X_{NM}$. For any $(i_1, j_1)$ and $(i_2, j_2)$ in $X_{NM}$, we assume $X_{NM}$ meets the following three conditions [9, 10].

1) $i_1 = i_2$ if and only if $j_1 = j_2$

2) $i_1 < i_2$ if and only if $j_1 < j_2$

3) $i_1$ is ancestor of $i_2$ if and only if $j_1$ is an ancestor of $j_2$

The above three conditions preserve local structure within a mapping $X_{NM}$. However, as illustrated in Figure 3, our concern is change of software architecture. So, the condition 3) is modified to 3a) as follows

3a) For $i_1$, $i_2$, there is a sub-tree $Si_1$ and $Si_2$ and an element e(i) of maximum ordering number starting from i,

$$e(i_1) < i_2 \text{ if and only if } e(Si_1) < Si_2$$

If a mapping, $X_{NM}$, meets the conditions of 1), 2) and 3a), it is generally called *structure preservation mapping*. By setting the origin of the sub-tree to the root of the tree, we call the sub-tree, meeting the conditions of 1), 2) and 3a), as *APM* (*Architecture Preservation Mapping*).

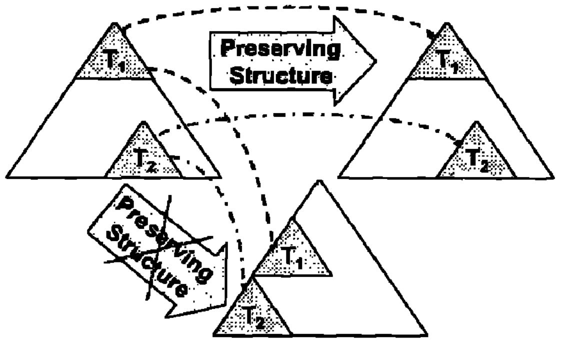

**Figure 3 Preserving Structure of Software Architecture**

Now, we can define *Architecture Weight* which measures the size of architecture. From Figure 3, we assume an element at lower level of a tree is less influential to the architecture evolution than that at higher level of tree. So, we define the total size of the architecture by the weighted sum of the nodes. And, assume the weight is inverse proportional to the depth of the tree. Thus, the architecture weight, $W_0$, is defined by equation (2).

$$W_0 = \sum_d (1/2^d)n(d) / \sum_d n(d) \qquad (2)$$

Where $d \geq 0$ denotes the depth at the tree, and n(d) is the number of the nodes at the depth. Note that $\sum n(d)$ is total number of components, such as classes or modules.

Base on the equation (2), we can calculate the weight of preserved architecture, Wp, by equation (3).

$$Wp = \sum_d (1/2^d)n_p(d) / \sum_d n_p(d) \qquad (3)$$

Where, $n_p(i)$ denotes number of nodes within the APM at the depth $d \geq 0$.

We define a metrics of *APF* (*Architecture Preservation Factor*) by equation (4).

$$APF = Wp/W_0 \qquad (4)$$

Note $0 \leq APF \leq 1.0$ and $W_0$ is evaluated for the *evolved* system.

(3) Numerical Example

For the example illustrated in Figure 2;

$W_0 = 1 + 2*(1/2) + 3*(1/4) + 3*(1/8) = 3.125$

$Wp = 1 + 2*(1/2) + 2*(1/4) + 2*(1/8) = 2.750$

APF =0.88

If we calculate the APF without the depth weight, $W_0$, Wp and APF could be 9.0, 7.0 and 0.77, respectively.

So, the APF with depth weight reflects that the changes at the higher level of hierarchy could cause wider and more complicated influence on the elements through the hierarchical structure. Thus, the cost of architectural evolution is *non-linear*.

## 2.3 Architecture Preservation Core and Cost of Architecture Evolution

(1) Architecture Preservation Core

Based on the concept of architecture preservation, a question arises how much architecture can be preserved over certain period of successive evolution as illustrated in Figure 4. Thus, we propose a concept of *APC (Architecture Preservation Core)* which is preserved part in the architecture during the period of evolution. For a sequence of successive evolution from 1 to i, $APC_i$ is defined as a union of a sequence of architecture preservation mappings defined by equation (5).

$$APC_i = APC_{i-1} \cap X_{i-1,i} = X_{12} \cap X_{23} \cap \quad \cap X_{i-1,i} \qquad (5)$$

Note that $APC_i \subseteq T_n$ for all n such that $1 \leq n \leq i$.

(2) APC and Architecture Reengineering

As defined by equation (5), $APC_i$ actually changes along with evolution. Since the evolution is open-ended process, it's unable to define minimal $APC_i$ against any unseen changes over infinite evolution period. However, an $APC_i$ represent the core of software architecture preserved until i-th evolution. Thus, we can estimate $APC_i$ from the trace of evolution history.

APC has an important meaning for architecture design. For example, APC can be interpreted as *frozen spot* in application frameworks.

We expect $APC_i$ plays an important criterion to identify whether an evolutional change needs to change core architecture. If any evolution needs to change $APC_i$, it may cause a major change over the entire software architecture, that is, discontinuous evolution.

For example, we observed a reengineering of platform architecture as a discontinuous evolution in the evolution of mobile phone product line discussed later.

(3) Cost of Architecture Evolution

The cost function defined by equation (1) can be revised by means of architecture preservation cost.

The discussions on architecture preservation core lead to that the

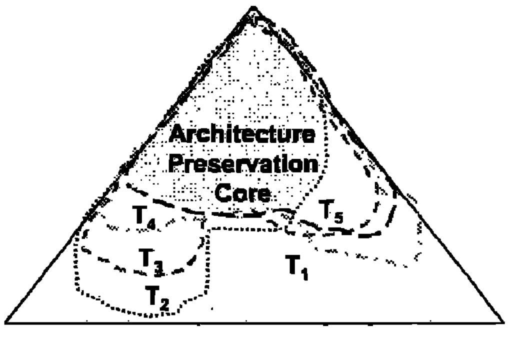

**Figure 4 Architecture Preservation over the Evolutions**

cost of changing architecture core will be much higher than that of other parts. So, the cost equation (1) can be revised to equation (6) by considering the architecture weights defined by equation (2).

$$C_l = (1/2^d)[W_A*\sum_i A_i^d + W_R*\sum_j R_j^d + W_M*\sum_k M_k^d] \quad (6)$$

Note that $A_i^d$, $R_j^d$ and $M_k^d$ respectively denotes the cost of Add, Remove, and Modify operation at depth d.

## 3. Evolution Dynamics

### 3.1 Continuous and Discontinuous Evolution

The changes, required to outside architecture preservation core at the lower levels of hierarchy tree, will create continuous, or incremental, evolution. However, if the changes invade the architecture preservation core and demand architecture redesign, the amount of changes required will be much higher. So, it triggers discontinuous evolution.

It should be remind that discontinuous evolution is not necessary a bad thing. For adapting the discontinuous change of requirements, the discontinuous evolution will be necessary. An example is the evolution of mobile phone software systems across multiple generations of base technologies illustrated in Figure 5 [3]. We observed two discontinuous evolutions for adapting the fundamental change of base technologies. This phenomenon can be considered as a correspondence to *adaptive mutation* in genetic evolution.

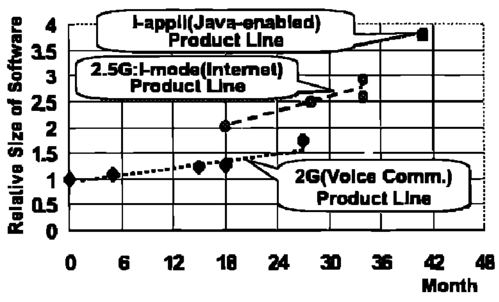

Figure 5 Software Evolution of Mobile Phone Software Systems

### 3.2 Architecture Design for Evolution

The concept of architecture preservation core implies that the architecture core should be coherent and compact. This is conventional wisdom in architecture design.

An important implication of the above discussion is that evolution process can help to identify the APC. As illustrated in Figure 6, the zone around the APC can be repeatedly changed over the evolution. So, such part of a system can be called *active evolution zone*. In application frameworks, it should be designed as *hot spot*. By exploring the evolution history, we may systematically identify the APC as well as active evolution zone.

### 3.3 Evolution Dynamics Revisited with Discontinuity

The discussions in the previous chapter lead to that evolution dynamics comprises of continuous and discontinuous evolution. And, the emergence of discontinuous evolution is much less frequent than that of continuous evolution.

From software architecture point of view, discontinuous evolution can be interpreted as follows:

1) Structure: Major reengineering of software architecture, and
2) Economics: High cost of evolution.

So, we can claim that evolution dynamics is *non-homogeneous along with time*.

We can also claim that evolution dynamics is *non-homogeneous across the space*, that is, part of a system. The cost of the evolution varies depending on the parts changed. Taking into account of the architecture preservation core, the cost model implies that evolution cost will be lower for the change at the lower levels of hierarchy, that is, at the *peripherals* of software architecture.

Conventional experience in software evolution suggests locality of the parts repeatedly changed over time. For example, empirical study of five different business applications found that 18.8% of the modules in the systems are repeatedly changed over their entire lifetimes. These experience leads to the notion of active evolution zone, hence non-homogeneous evolution across the software systems.

## 4. Conclusions

In this article, we proposed a set of metrics for software architecture evolution, and discussed continuity and discontinuity in software evolution with the metrics proposed. From our experience, discontinuous evolution plays as a central mechanism to adopt the software systems to the essential changes of requirements as well as computing environments. However, it could be also a big stress to the software development. Thus, we need a design methodology for ease of evolution. Further study on the continuous and discontinuous evolutions will be necessary.

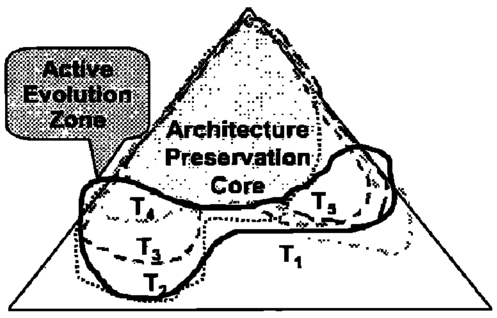

Figure 6 Active Evolution Zone

## References

[1] A. I. Anton and C. Potts, Functional Paleontology: System Function as the User Sees It, *Proc. ICSE 2001*, May 2001, pp. 421 - 430.

[2] M. Aoyama, Evolutionary Patterns of Design and Design Patterns, *Proc. ISPSE 2000*, IEEE Computer Society Press, Nov. 2000, pp. 110-116.

[3] M. Aoyama, Continuous and Discontinuous Software Evolution: Aspects of Software Evolution across Multiple Product Lines, *Proc. IWPSE 2001*, ACM Press, Sep. 2001.

[4] N. Eldredge, *The Pattern of Evolution*, W. H. Freeman and Company, 1999.

[5] M. Godfrey and Q. Tu, Growth, Evolution, and Structural Change in Open Source Software, *Proc. IWPSE 2001*, ACM Press, Sep. 2001.

[6] T. Katayama, Evolutionary Domains: A Basis for Sound Software Evolution, *Proc. IWPSE 2001*, ACM Press, Sep. 2001.

[7] C. F. Kemerer and S. A. Slaugther, A Longitudinal Empirical Analysis of Software Evolution, *Proc. IWPSE '98*, May 1998, pp. 21-28.

[8] M. M. Lehman and J. Ramil, Evolution in Software and Related Areas, *Proc. IWPSE 2001*, ACM Press, Sep. 2001.

[9] K. C. Tai, The Tree-to-Tree Correcting Problem, *J of ACM*, Vol. 26, 1979, pp. 422-433.

[10] E. Tanaka, Structural Distance and Similarities, *J. IPSJ*, Vol. 31, No. 9, Sep. 1990, pp. 1270-1279 (In Japanese).