ソフトウェア工学　73－1
（１９９０．７．１９）

ソフトウェアの構成的変化に関する理論的分析

大森　晃
富士通㈱国際情報社会科学研究所

青山　幹雄
富士通㈱複合交換機事業部

概　要

ソフトウェア・システムはファミリーとして構成的に変化・進展する．ソフトウェアの構成的側面およびその変化を記述し管理するための技法として，ソフトウェア構成管理が適用されている．しかし，これまでの構成管理技法には理論的な基礎が欠如している．構成的に変化・進展するソフトウェア・システム・ファミリーを管理するためには，しっかりとした，また見通しのよい理論的枠組みが不可欠である．この論文では，そうした枠組みに向けての第一歩として，ソフトウェア・システム構成を形式化し，その構成的変化を理論的に分析している．また，理論的結果をソフトウェアの構成的変化の分類に適用し，分類上の複雑さを大幅に減少しうることを示している．

Theoretical Analyses of Software Architectural Changes

*Akira OHMORI * and Mikio AOYAMA ***

** International Institute for Advanced Study of Social Information Science*
Fujitsu Limited, 17-25, Shinkamata 1-Chome, Ota-ku, Tokyo 144, JAPAN.
*** Telecommunications Software Division*
Fujitsu Limited, 629 Shimo-Kodanaka, Nakahara-Ku, Kawasaki 211, JAPAN.

ABSTRACT

Software systems architecturally change or evolve as a family. Software configuration management (SCM) is an emerging technique used to describe and manage software architectural aspects and their changes. However, conventional SCM techniques lack theoretical bases. A sound and penetrating theoretical framework is indispensable to manage an architecturally evolving family of software systems. As a step toward such a framework, in this paper we formalize a software system configuration and theoretically analyze its architectural changes. Our theoretical analysis is applied to a taxonomy of software architectural changes. Through this application, we show that combinatorial complexity of change categories can be drastically reduced.

1. Introduction

A software system architecturally changes or evolves during development and maintenance. Software configuration management (SCM) addresses such changes from a variety of viewpoints. Conventional studies focus on the methodology and environment of SCM [Bers80, Nara87, Yau87, Baze85] . These works do not explicitly address the analysis of software architectural changes, which is concerned with both elemental changes and structural changes made to software. The importance of analysis of software architectural changes to be managed is no wonder for SCM.

Software systems tend to form a family. We have observed the evolution and architectural changes of a family of switching software systems as illustrated in Fig. 1. The family consists of several models which fulfill different requirements to the performance and capacity. However, they are developed based on a single design concept. Numerous software components are used in common by different models, which evolve in parallel and interactively with one another. As they evolve, new models are derived and step in their respective evolutional ways. In this way, switching software systems evolve as a family.

The family concept is effective to the productivity and quality of switching software systems. However, the maintenance and management of such an evolving family are error-prone tasks due to a large number of software components and complicated relationships among them, and also due to complicated interrelations among systems in the family. Yet, conventional SCM does not cover such aspects.

The idea underlying SCM is promising to manage the evolution of such a complicated family. However, the theory of SCM is still in its infancy, and so the management methods tend to be ad hoc, depending on the application domain. A sound and penetrating theoretical framework of SCM is indispensable to effectively establish management systems of family-widely evolving software systems.

Thus, as a step toward such a framework, this paper analyzes software architectural changes in a theoretical manner with an entity and relationship approach. In Sections 2 and 3, we formalize a software system configuration and set-theoretically analyze its architectural changes. Each theoretical result is presented as a proposition, and their proofs are outlined in Appendix. Section 4 shows that, based on our theoretical framework, categorical complexity of changes can be drastically reduced in a taxonomy of software architectural changes.

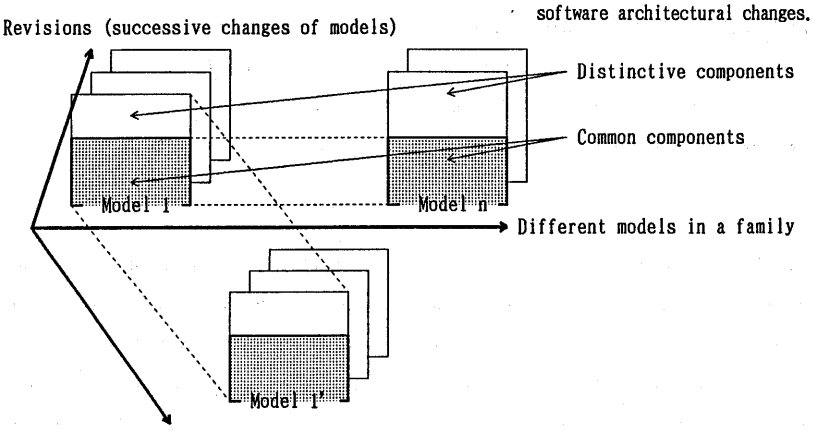

Fig. 1 A family of software architectural changes

2. Software System Configurations

A software system configuration can be described in terms of software architectural entity and relationship between entities [McCa75, Bers80, Nara87] . In this section, we first introduce the concept of a software architectural entity, and a relationship between entities. And then, we describe a software system configuration.

2.1 Software Architectural Entities

A functional unit in a software system is called a software architectural entity, or simply an entity. We can recognize three types of software architectural entity as follows.

(1) Module Entities

A module entity is a minimal software architectural entity to be developed and managed.

(2) Feature Entities

There can be a software architectural entity which uses one or more module entities in order to implement its function. Furthermore, there can be a software architectural entity which uses such entities; and so on. The entity which is of this type, and used by other entities, is called a feature entity. A feature entity architecturally lies between a module entity and a system entity which we next mention. It is often called a program or a subsystem in practice.

(3) System Entities

A system entity is a software architectural entity which uses one or more feature entities in order to implement its function and can be regarded as a delivery unit to a customer.

The form of a software architectural entity is not restricted to an executable one. Even if a software architectural entity is nonexecutable, it is not restricted to a set of statements written in a programming language, either. It can be a requirements specification, or a functional specification, or a logical specification.

We observe an entity from a macroscopic viewpoint, and concentrate on its name and body. Thus, any entity e is expressed as a pair of its name n and its body b, that is, $e=(n,b)$. This formalism is motivated by management experiences in practical software development. If we consider a program as an entity $e=(n,b)$, n and b respectively correspond to its name and the set of its statements.

We assume that there is a universal set E that contains all software architectural entities.

The name of a system entity might be different from the name of a software system that contains the system entity because the system entity is not a software system itself. In this paper, however, the name of a system entity will be regarded as that of a software system, and vice versa.

2.2 Entity-Relationships

In the previous section, we have mentioned that an entity can 'use' other entities. From this observation, we introduce a binary relation USE in the set E as follows:

$(e,e')\in USE \subset E\times E$

$\Leftrightarrow$ 'the entity e uses the entity e' in order to implement its function.'

For a USE relation, we assume the followings.

(1) Antireflexive-use assumption

For any $e\in E$, $(e,e)\notin USE$.

(2) Asymmetric-use assumption

For any e and $e'\in E$, if $(e,e')\in USE$, $(e',e)\notin USE$.

(3) Direct-use assumption

$USE \cap (USE \cdot USE)=\phi$, where $USE \cdot USE$ is the composition of USE relations.

The assumption (1) implies that any entity does not externally use itself. Thus, it allows a kind of reflexive relationship, such as recursion mechanism, which can be regarded as an internal relationship. Such a relationship is not explicitly addressed in our framework. The assumption (2) implies that if an entity e uses another entity e', the entity e' never uses the entity e. If an entity e uses another entity e' and the

entity e' uses another entity e'', it might be considered that the entity e uses the entity e''. The assumption (3) implies that such a relationship does not exist in a *USE* relation.

2.3 System Configurations

In this subsection, we describe a software system configuration based on the entity set E and the entity-relationship *USE*.

Definition 1

Let T be a time set and $t \in T$ be arbitrary; s any software system name. Let $E(s;t) \subset E$ be the set of entities which are employed at time t to configure a specific software system named s. Let $USE/E(s;t)$ be the relation induced by *USE* in $E(s;t)$, that is, $USE/E(s;t) = USE \cap (E(s;t) \times E(s;t))$.

Then, if $SC(s;t) = \langle E(s;t); USE/E(s;t) \rangle$ satisfies the following conditions, it is called a system configuration for s at time t.

(a) a single system entity

$E(s;t)$ contains one, and only one system entity named s.

(b) a finite entity set

$E(s;t)$ is a finite set.

(c) a single name for the same body

For any (n,b) and $(n',b') \in E(s;t)$, if $b=b'$, then $n=n'$.

(d) a single body for the same name

For any (n,b) and $(n',b') \in E(s;t)$, if $n=n'$, then $b=b'$.

(e) no isolated entity

If $Card(E(s;t)) \geq 2$, for any entity $e \in E(s;t)$, there exits an entity $e' \in E(s;t)$ such that $(e,e') \in USE$ or $(e',e) \in USE$, where $Card(E(s;t))$ is the cardinal number of $E(s;t)$, that is, the number of entities in $E(s;t)$. □

A software system configuration consists of the two parts, that is, the elemental part $E(s;t)$ and the structural part $USE/E(s;t)$. We can easily show that the structural part inherits the assumptions to a *USE* relation.

In practice, there can be an isolated entity; several different bodies can have the same name; and there can be a body which has some different names. Those conditions (c), (d) and (e), which can be regarded as desirable conditions for a software system configuration, are introduced to avoid such managerial inconvenience and obtain a sound theoretical base.

2.4 Preliminary Analysis of A System Configuration

We have some preliminary analytic results concerned with a system configuration $SC(s;t) = \langle E(s;t); USE/E(s;t) \rangle$. In order to show them, let us introduce the following notations.

$$NAME(SC(s;t)) = \{ n \mid (n,b) \in E(s;t) \} .$$

$$BODY(SC(s;t)) = \{ b \mid (n,b) \in E(s;t) \} .$$

$$N\text{-}struct(SC(s;t)) = \{(n,n') \mid ((n,b),(n',b')) \in USE/E(s;t)\} .$$

$$B\text{-}struct(SC(s;t)) = \{(b,b') \mid ((n,b),(n',b')) \in USE/E(s;t)\} .$$

$NAME(SC(s;t))$ and $N\text{-}struct(SC(s;t))$ represent the nominal aspects of a system configuration $SC(s;t) = \langle E(s;t); USE/E(s;t) \rangle$; on the other hand, $BODY(SC(s;t))$ and $B\text{-}struct(SC(s;t))$ represent the bodily aspects. These notations are illustrated in Fig. 2, where n_i and b_i are respectively an entity name and an entity body.

Proposition 1

For any system configuration $SC(s;t) = \langle E(s;t); USE/E(s;t) \rangle$, $Card(E(s;t)) = Card(NAME(SC(s;t))) = Card(BODY(SC(s;t)))$. □

Proposition 1 analyzes a relationship among the number of entities, the number of names and the number of bodies used for the elemental part of a system configuration. And it shows that those numbers are the same. This result seems to be trivial. However, if we do not have the conditions of 'a single name for the same body' and 'a single body for the same name', which have been mentioned in the previous section, the above result cannot be derived.

Proposition 2

Let $SC(s;t) = \langle E(s;t); USE/E(s;t) \rangle$ be an arbitrary system configuration. If $Card(E(s;t)) = 1$, then $USE/E(s;t) = \phi$ and therefore $N\text{-}struct(SC(s;t)) = \phi$ and $B\text{-}struct(SC(s;t)) = \phi$. □

Proposition 3

Let $SC(s;t) = \langle E(s;t); USE/E(s;t) \rangle$ be an arbitrary system configuration. If $Card(E(s;t)) \geqq 2$, then $USE/E(s;t) \neq \phi$ and therefore $N\text{-}struct(SC(s;t)) \neq \phi$ and $B\text{-}struct(SC(s;t)) \neq \phi$. □

Propositions 2 and 3 show that whether the structural part of a system configuration is an empty or not depends on the number of entities employed. From Proposition 2, if a system configuration has only one entity, its structural part is an empty. For this result, the antireflexive-use assumption mentioned in Section 2.2 is critical. On the other hand, Proposition 3 means that if a system configuration has at least two entities, its structural part is not an empty. The condition of 'no isolated entity' mentioned in the previous section is critical for this result.

3. Analysis of Software Architectural Changes

In this section, we show analytic results concerned with the differences between two system configurations.

Proposition 4

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations and $E(s;t) \neq E(s';t')$. Then, the following hold.

(a) If $Card(E(s;t)) = 1$ and $Card(E(s';t')) = 1$, $USE/E(s;t) = USE/E(s';t')$.

(b) If $Card(E(s;t)) = 1$ and $Card(E(s';t')) \geqq 2$, $USE/E(s;t) \neq USE/E(s';t')$.

(c) If $Card(E(s;t)) \geqq 2$ and $Card(E(s';t')) = 1$, $USE/E(s;t) \neq USE/E(s';t')$.

(d) If $Card(E(s;t)) \geqq 2$ and $Card(E(s';t')) \geqq 2$, $USE/E(s;t) \neq USE/E(s';t')$. □

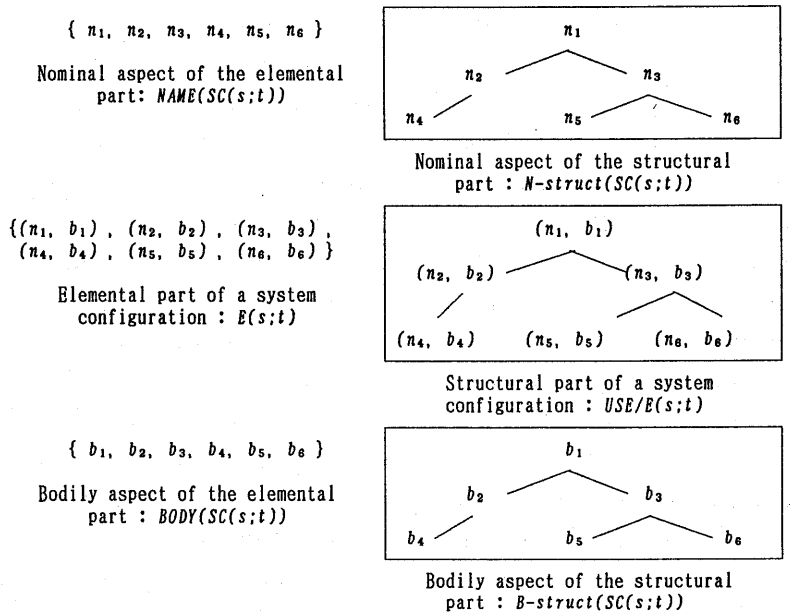

Fig. 2 The nominal and the bodily aspect of a system configuration

Proposition 4 addresses two system configurations whose elemental parts are different from each other. And it shows how the cardinal numbers of their elemental parts influence a relationship between their structural parts. The condition of 'no isolated entity' mentioned in Section 2.3 explicitly plays an important role in Proposition 4, and also does in Propositions 10, 11 and 15 shown in what follows.

From Proposition 4, we have the following three propositions, that is, Propositions 5, 6 and 7. For any two system configurations, these propositions show that the situation of their elemental parts and structural parts is closely related to the cardinal numbers of their elemental parts.

Proposition 5

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. If $E(s;t) \neq E(s';t')$ and $USE/E(s;t) = USE/E(s';t')$, then $Card(E(s;t))=1$ and $Card(E(s';t'))=1$.□

Proposition 6

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. If $E(s;t) \neq E(s';t')$ and $USE/E(s;t) \neq USE/E(s';t')$, then $Card(E(s;t))=1$ and $Card(E(s';t'))=1$ do not hold.□

Proposition 7

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. If $E(s;t) \neq E(s';t')$, $USE/E(s;t) \neq USE/E(s';t')$ and $Card(E(s;t))= Card(E(s';t'))$, then $Card(E(s;t)) \geq 2$ and $Card(E(s';t')) \geq 2$.□

Up to this point, we have analyzed software architectural changes without distinction of entity name and entity body. In what follows, we explicitly consider the nominal and the bodily aspect of a system configuration.

At first, we have the following proposition. It shows some implications of the case where the elemental parts in any two system configurations are not different at all, in other words, the case where any software architectural change does not occur.

Proposition 8

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations; and $E(s;t)=E(s';t')$. Then, the following hold.

(a) $Card(E(s;t))= Card(E(s';t'))$.

(b) $NAME(SC(s;t)) = NAME(SC(s';t'))$.

(c) $BODY(SC(s;t)) = BODY(SC(s';t'))$.

(d) $USE/E(s;t)= USE/E(s';t')$.

(e) $N\text{-}struct(SC(s;t)) = N\text{-}struct(SC(s';t'))$.

(f) $B\text{-}struct(SC(s;t)) = B\text{-}struct(SC(s';t'))$. □

We encounter the following situation in the nominal or bodily aspect of a system configuration when its elemental part only changes.

Proposition 9

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. If $E(s;t) \neq E(s';t')$ and $USE/E(s;t) = USE/E(s';t')$, then $NAME(SC(s;t)) \neq NAME(SC(s';t'))$ or $BODY(SC(s;t)) \neq BODY(SC(s';t'))$.□

For any two system configurations, the following two propositions analyze their nominal aspects and bodily aspects, respectively, under the assumption that each of the system configurations has at least two entities. Proposition 10 shows that if the nominal aspects of the structural parts are the same, the nominal aspects of their elemental parts are also the same. Proposition 11 shows a similar result from a bodily point of view.

Proposition 10

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations; $Card(E(s;t)) \geq 2$; $Card(E(s';t')) \geq 2$; and $N\text{-}struct(SC(s;t))= N\text{-}struct(SC(s';t'))$. Then, $NAME(SC(s;t))=NAME(SC(s';t'))$.□

Proposition 11

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations; $Card(E(s;t)) \geq 2$; $Card(E(s';t')) \geq 2$; and $B\text{-}struct(SC(s;t))= B\text{-}struct(SC(s';t'))$. Then, $BODY(SC(s;t))=BODY(SC(s';t'))$.□

In the above two propositions, the assumption of $Card(E(s;t)) \geq 2$ and $Card(E(s';t')) \geq 2$ plays an important role. If such an assumption does not hold, we

have the following two propositions which are respectively similar to but weaker than Proposition 10 and Proposition 11.

Proposition 12

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. If $N\text{-}struct(SC(s;t))=N\text{-}struct(SC(s';t'))$, then $Card(E(s;t))=Card(E(s';t'))$. □

Proposition 13

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. If $B\text{-}struct(SC(s;t))=B\text{-}struct(SC(s';t'))$, then $Card(E(s;t))=Card(E(s';t'))$. □

For any two system configurations, we can consider several differences between their nominal aspects, and their bodily aspects, more concretely than a mere inequality. In the following propositions, we address two differences.

Proposition 14

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations. Then, the following hold.

(a) If $NAME(SC(s;t))\cap NAME(SC(s';t'))=\phi$, then $N\text{-}struct(SC(s;t))\cap N\text{-}struct(SC(s';t'))=\phi$.

(b) If $BODY(SC(s;t))\cap BODY(SC(s';t'))=\phi$, then $B\text{-}struct(SC(s;t))\cap B\text{-}struct(SC(s';t'))=\phi$. □

Proposition 14 (a) shows that, for any two system configurations, if they do not share the same entity name at all, the nominal aspects of their structural parts are completely different. This situation corresponds to the case where the entity names are all replaced by other entity names. Proposition 14 (b) shows a similar result from a bodily point of view.

Proposition 15

Let $SC(s;t)$ and $SC(s';t')$ be arbitrary system configurations; $Card(E(s;t))\geqq 2$. Then, the following hold.

(a) If $N\text{-}struct(SC(s;t))\subset N\text{-}struct(SC(s';t'))$, then $NAME(SC(s;t))\subset NAME(SC(s';t'))$.

(b) If $B\text{-}struct(SC(s;t))\subset B\text{-}struct(SC(s';t'))$, then $BODY(SC(s;t))\subset BODY(SC(s';t'))$.□

Proposition 15 shows that, for any two system configurations, if one is structurally included in the other, the former is also elementally included in the latter. The case (a) addresses the nominal aspect of a system configuration, on the other hand, the case (b) does the bodily aspect. This proposition corresponds to the structural and elemental enlargement of a system configuration.

4. A Taxonomy of Software Architectural Changes

In this section, we discuss what viewpoints can be used for a taxonomy of software architectural changes, and how many categorical candidates of changes we can encounter. Furthermore, we describe how our theoretical analysis can be used for reducing the categorical complexity.

4.1 Taxonomical Viewpoints

Let us consider the case where a system configuration $SC(s;t)=\langle E(s;t);USE/E(s;t)\rangle$ changes to a system configuration $SC(s';t')=\langle E(s';t');USE/E(s';t')\rangle$. Then, we have four types of changes, including invariant, as follows.

(A) $E(s;t)=E(s';t')$ and $USE/E(s;t)=USE/E(s';t')$.

(B) $E(s;t)=E(s';t')$ and $USE/E(s;t)\neq USE/E(s';t')$.

(C) $E(s;t)\neq E(s';t')$ and $USE/E(s;t)=USE/E(s';t')$.

(D) $E(s;t)\neq E(s';t')$ and $USE/E(s;t)\neq USE/E(s';t')$.

The elemental parts $E(s;t)$ and $E(s';t')$ have their nominal aspects and bodily aspects, and so do the structural parts $USE/E(s;t)$ and $USE/E(s';t')$. For each type of change, therefore, we can consider sixteen types of changes based on the following four cases:

(1) whether or not $NAME(SC(s;t))\neq NAME(SC(s';t'))$,

(2) whether or not $BODY(SC(s;t))\neq BODY(SC(s';t'))$,

(3) whether or not $N\text{-}struct(SC(s;t))\neq N\text{-}struct(SC(s';t'))$, and

(4) whether or not $B\text{-}struct(SC(s;t))\neq B\text{-}struct(SC(s';t'))$.

At this point, we have 64 categories of changes, including 60 categories for which we can observe some change in at least one aspect of the system configuration. From more detailed observations, we have

16 categories where only one aspect of the system configuration changes, 24 categories for two aspects, 16 categories for three aspects, and 4 categories for four aspects.

In the above discussions, the difference between $SC(s;t)$ and $SC(s';t')$ is denoted by a mere inequality. We consider such a difference more concretely in the following.

Let us consider, for example, $NAME(SC(s;t)) \neq NAME(SC(s';t'))$. We can classify this difference into the following three cases.

(a) $NAME(SC(s;t)) \cap NAME(SC(s';t')) = \phi$.

(b) $NAME(SC(s;t)) \subsetneq NAME(SC(s';t'))$ or $NAME(SC(s';t')) \subsetneq NAME(SC(s;t))$.

(c) $\neg(NAME(SC(s;t)) \subsetneq NAME(SC(s';t')))$ and $\neg(NAME(SC(s';t')) \subsetneq NAME(SC(s;t)))$ and $NAME(SC(s;t)) \cap NAME(SC(s';t')) \neq \phi$, where $\neg($ a statement $)$ means the negation of the statement.

For the other three aspects, their respective differences can be similarly classified into three cases, also.

At this point, we have 1,024 categories of changes, which is derived from the algebraic expression $(4 \times 3^0) + (16 \times 3^1) + (24 \times 3^2) + (16 \times 3^3) + (4 \times 3^4)$. However, we can investigate whether or not each of those candidates is theoretically meaningful. In that case, the theoretical analysis in Sections 2 and 3 plays an important role for reducing the categorical complexity of changes. In order to examine it, we show some application examples in the next section.

4.2 Application of Theoretical Analysis to Taxonomy of Changes

Example 1 : Consider the case where $E(s;t)=E(s';t')$ and $USE/E(s;t) = USE/E(s';t')$. From Proposition 8, we have that $NAME(SC(s;t)) = NAME(SC(s';t'))$, $BODY(SC(s;t)) = BODY(SC(s';t'))$, $N\text{-}struct(SC(s;t)) = N\text{-}struct(SC(s';t'))$ and $B\text{-}struct(SC(s;t)) = B\text{-}struct(SC(s';t'))$. Therefore, we can eliminate 255 out of 256 change categories.□

Example 2 : Consider the case where $E(s;t)=E(s';t')$ and $USE/E(s;t) \neq USE/E(s';t')$. This type of change shows that the elemental part does not change but the structural part does. However, (d) of Proposition 8 does not allow such a type of change. Thus, all of 256 change categories are not feasible.

Example 3 : Consider the case where $E(s;t) \neq E(s';t')$ and $USE/E(s;t) = USE/E(s';t')$. From Propositions 5 and 2, we have that $N\text{-}struct(SC(s;t)) = N\text{-}struct(SC(s';t'))$ and $B\text{-}struct(SC(s;t)) = B\text{-}struct(SC(s';t'))$. From Proposition 9, we have that $NAME(SC(s;t)) \neq NAME(SC(s';t'))$ or $BODY(SC(s;t)) \neq BODY(SC(s';t'))$. Therefore, we can eliminate 241 out of 256 change categories. For such a type of change, note that we address a system configuration which consists of only a system entity. From this fact, we finally have 3 acceptable change categories.

Example 4 : Consider the case where $E(s;t) \neq E(s';t')$ and $USE/E(s;t) \neq USE/E(s';t')$. From Propositions 12, 7 and 10, we cannot allow any type of changes that satisfy both $N\text{-}struct(SC(s;t)) = N\text{-}struct(SC(s';t'))$ and $NAME(SC(s;t)) \neq NAME(SC(s';t'))$. Similarly, from Propositions 13, 7 and 11, we cannot allow any type of changes that satisfy both $B\text{-}struct(SC(s;t)) = B\text{-}struct(SC(s';t'))$ and $BODY(SC(s;t)) \neq BODY(SC(s';t'))$. These results enable us to eliminate 87 out of 256 change categories.

Example 5 : Consider the case where $E(s;t) \neq E(s';t')$ and $USE/E(s;t) \neq USE/E(s';t')$ once more. And let $NAME(SC(s;t)) \neq NAME(SC(s';t'))$; $BODY(SC(s;t)) \neq BODY(SC(s';t'))$; $N\text{-}struct(SC(s;t)) \neq N\text{-}struct(SC(s';t'))$; and $B\text{-}struct(SC(s;t)) \neq B\text{-}struct(SC(s';t'))$. In this case, we have 81 change categories. However, we can eliminate 45 change categories with Propositions 14 and 15. For example, let $NAME(SC(s;t)) \cap NAME(SC(s';t')) = \phi$ and $BODY(SC(s;t)) \cap BODY(SC(s';t')) = \phi$. Then, from Proposition 14, we can only accept the combination of $N\text{-}struct(SC(s;t)) \cap N\text{-}struct(SC(s';t')) = \phi$ and $B\text{-}struct(SC(s;t)) \cap B\text{-}struct(SC(s';t')) = \phi$. Therefore, we can eliminate 8 out of 9 change categories.

Through similar applications to other cases, we can furthermore eliminate 48 change categories. Therefore, we can conclude that our theoretical analysis makes it possible to eliminate 944 out of 1,024 change categories.

5. Conclusions

From our practical experiences, we have pointed out, that a theoretical framework of software configuration management is indispensable to manage a family of evolving software systems. As a step toward such a framework, we have theoretically analyzed architectural changes of software configurations from both an elemental and a structural aspect. It provides a penetrating framework and leads to a better position, to have an insight into software architectural changes. Indeed, we have applied our theoretical analysis to a taxonomical study of software architectural changes, and have shown that combinatorial complexity of change categories can be drastically reduced.

In the future, it is necessary to explicitly study family-wide architectural changes of software configurations through an extension of our present framework. Furthermore, it is also important to study quality aspects of software family evolution in terms of configuration. These studies will lead to an integrated framework of SCM and software quality management [Aoya88, Ohmo89].

References

[Aoya88] M. Aoyama, Y. Hanai and M. Suzuki(1988):"An Integrated Software maintenance Environment", Proc. IEEE Conf. on Software Maintenance, Oct., Phoenix, pp.40-44.

[Baze85] R. Bazelmans(1985): "Evolution of Configuration Management", ACM SIGSOFT Software Engineering Notes, Vol.10, No.5, Oct., pp.37-46.

[Bers80] E. H. Bersoff, V. D. Henderson and S. G. Siegel(1980): Software Configuration Management, Prentice-Hall.

[McCa75] R. McCarthy(1975): "Applying the Technique of Configuration Management to Software", Quality Progress, Oct.

[Nara87] K. Narayanaswamy and W. Scacchi(1987): "Maintaining Configurations of Evolving Software Systems", IEEE Trans. on Software Engineering, Vol.SE-13, No.3, pp. 324-334.

[Ohmo89] A. Ohmori(1989):"Quality Deployment Method in Software Development", Engineers, Nikka-Giren, No.485, Feb., pp.6-10.

[Yau 87] S. S. Yau and J. J. Tsai(1987): "Knowledge Representation of Software Component Interconnection Information for Large-Scale Software Modifications", IEEE Trans. on Software Engineering, Vol.SE-13, No.3, pp.355-361.

Appendix

Proof of Proposition 1 : From the conditions of 'a single body for the same name' and 'a single name for the same body' for a system configuration, we can show that a bijection between $NAME(SC(s;t))$ and $E(s;t)$ exists. Similarly, we can also show that a bijection between $BODY(SC(s;t))$ and $E(s;t)$ exists. □

Proof of Proposition 2 : We have $USE/E(s;t) = \phi$ from the antireflexive-use assumption for a USE relation. This implies that $N\text{-}struct(SC(s;t)) = \phi$ and $B\text{-}struct(SC(s;t)) = \phi$. □

Proof of Proposition 3 : We have $USE/E(s;t) \neq \phi$ from the condition of 'no isolated entity' for a system configuration. This implies that $N\text{-}struct(SC(s;t)) \neq \phi$ and $B\text{-}struct(SC(s;t)) \neq \phi$. □

Proof of Proposition 4 : (a) can be proved by Proposition 2. (b) and (c) can be proved by Proposition 2 and Proposition 3. For (d), from the condition $E(s;t) \neq E(s';t')$, we can show at first that there exists an entity $e \in E(s;t)$ which is not included in $E(s';t')$. Then, from the condition of 'no isolated entity' for a system configuration, we have an entity $e' \in E(s;t)$ such that $(e,e') \in USE/E(s;t)$. Since $e \notin E(s';t')$, we can conclude that $(e,e') \notin USE/E(s';t')$. □

Proof of Proposition 5 : Assume that $Card(E(s;t)) = 1$ and $Card(E(s';t')) = 1$ do not hold. From Proposition 4, this assumption implies that $USE/E(s;t) \neq USE/E(s';t')$.

Thus, we have a contradiction. □

Proof of Proposition 6 : Assume that $Card(E(s;t)) = 1$ and $Card(E(s';t')) = 1$. Then, this proposition can be proved in a similar way to the proof of Proposition 5. □

Proof of Proposition 7 : From Proposition 6, $Card(E(s;t)) = 1$ and $Card(E(s;t)) = 1$ do not hold. Since $Card(E(s;t)) = Card(E(s';t'))$, we can conclude that $Card(E(s;t)) \geq 2$ and $Card(E(s';t')) \geq 2$. □

Proof of Proposition 8 : (a) is obvious from $E(s;t) = E(s';t')$. (b) can be proved by showing that any entity name $n \in NAME(SC(s;t))$ is included in $NAME(SC(s';t'))$ and that any entity name $n' \in NAME(SC(s';t'))$ is included in $NAME(SC(s;t))$. (c) can be proved in a similar way to the proof of (b). (d) is also obvious from $E(s;t) = E(s';t')$, which implies that (e) and (f) hold. □

Proof of Proposition 9 : Let $NAME(SC(s;t)) = NAME(SC(s';t'))$ and $BODY(SC(s;t)) = BODY(SC(s';t'))$. From Proposition 5, $Card(E(s;t)) = 1$ and $Card(E(s;t)) = 1$. Then, it follows that $E(s;t) = E(s';t')$. This result is contradictory to the condition $E(s;t) \neq E(s';t')$. □

Proof of Proposition 10 : From the condition of 'no isolated entity' for a system configuration, we can show the following.

(a) For any $(n,b) \in E(s;t)$, there is a $(n',b') \in E(s';t')$ such that $n = n'$.

(b) For any $(n',b') \in E(s';t')$, there is a $(n,b) \in E(s;t)$ such that $n' = n$.

From (a), we have that $NAME(SC(s;t)) \subset NAME(SC(s';t'))$. And from (b), we have that $NAME(SC(s';t')) \subset NAME(SC(s;t))$. □

Proof of Proposition 11 : From the condition of 'no isolated entity' for a system configuration, we can show the following.

(a) For any $(n,b) \in E(s;t)$, there is a $(n',b') \in E(s';t')$ such that $b = b'$.

(b) For any $(n',b') \in E(s';t')$, there is a $(n,b) \in E(s;t)$ such that $b' = b$.

From (a), we have that $BODY(SC(s;t)) \subset BODY(SC(s';t'))$. And from (b), we have that $BODY(SC(s';t')) \subset BODY(SC(s;t))$. □

Proof of Proposition 12 : We consider the following cases:

(a) $Card(E(s;t)) = 1$ and $Card(E(s';t')) = 1$;

(b) $Card(E(s;t)) = 1$ and $Card(E(s';t')) \geq 2$;

(c) $Card(E(s;t)) \geq 2$ and $Card(E(s';t')) = 1$;

(d) $Card(E(s;t)) \geq 2$ and $Card(E(s';t')) \geq 2$.

In case (a), it is obvious that $Card(E(s;t)) = Card(E(s';t'))$. In case (b), since $N\text{-}struct(SC(s;t)) = \phi$ and $N\text{-}struct(SC(s';t')) \neq \phi$ by Proposition 2 and Proposition 3, we have a contradiction. In case (c), we have also the same result. In case (d), $Card(E(s;t)) = Card(E(s';t'))$ can be proved by Proposition 1 and Proposition 10. □

Proof of Proposition 13 : This proposition can be proved, in a similar way to the proof of Proposition 12, by Propositions 2 and 3, and by Propositions 1 and 11. □

Proof of Proposition 14 : Assume that $N\text{-}struct(SC(s;t)) \cap N\text{-}struct(SC(s';t')) \neq \phi$. Then, there exists a pair of entity names (n,n') such that $(n,n') \in N\text{-}struct(SC(s;t))$ and $(n,n') \in N\text{-}struct(SC(s';t'))$. From this, we can show that there exist entity bodies b and b' such that $(n,b) \in E(s;t)$ and $(n,b') \in E(s';t')$. Therefore, $n \in NAME(SC(s;t))$ and $n \in NAME(SC(s';t'))$, and so the contraposition of (a) holds. This proves (a). We can prove (b) in a similar way to the proof of (a). □

Proof of Proposition 15 : Take any $n \in NAME(SC(s;t))$. Then, there exists some entity body b such that $(n,b) \in E(s;t)$. Since $Card(E(s;t)) \geq 2$, from the condition of 'no isolated entity' for a system configuration, we can show that there exists some entity name n' such that $(n,n') \in N\text{-}struct(SC(s;t))$. Since $N\text{-}struct(SC(s;t)) \subset N\text{-}struct(SC(s';t'))$, $n \in NAME(SC(s';t'))$. This proves (a). (b) can be proved in a similar way to the proof of (a). □