MMRC
DISCUSSION PAPER SERIES

No. 333

複雑システムの設計進化:
ソフトウェアのアーキテクチャ変化の測定

兵庫県立大学
立本　博文

2010 年 12 月

東京大学ものづくり経営研究センター
Manufacturing Management Research Center (MMRC)

http://merc.e.u-tokyo.ac.jp/mmrc/dp/index.html

Design evolution in complex systems: An empirical study of the architectural changes during a series of software releases

Hirofumi Tatsumoto, Associate Professor

School of Business Administration, University of Hyogo

Dec. 2010

ABSTRACT

In complex systems, design elements have many dependencies that are classified into two types: dependencies within modules and dependencies among modules. Since dependencies among modules affect the division of labor more directly than those within modules, an architectural change, that is, the change in forms of dependencies among modules, makes big changes in the forms of division of labor.

Many case studies examined the impact of architectural changes through design evolution of complex systems. They demonstrated that an architectural change leads to structural change in the industry and causes established firms to be overwhelmed by new ones.

However, little is known about what actually happens in a complex system during architectural changes due to the lack of operational method and measurement of dynamic processes. This study develops an architecture index calculated from dependencies among modules to determine the pattern of dynamic process. Using the architecture indexes of 19 versions of a software application, we investigate the dynamic process during design evolution. As a result, we found three points as below.

i) The architecture of the software application does not change monotonously but swings back and forth between integral and modular during design evolution. These architectural swings are observed at any levels in the hierarchy of the software application. The previous studies emphasizing architectural change from integral straight to modular oversimplify the observed dynamic process.

ii) Many small and continuous changes happen at the lower levels while several big and unexpected changes happen at the upper levels. The difference of dynamic patterns between the lower and upper levels can cause the absence of organizational recognition: An organization that cares only about changes at lower level can ignore a big change at upper level.

iii) The architectural changes at the upper level correlate negatively with those at the lower level. For example, the architecture at the upper level goes to integral if the architecture at the lower level goes to modular. This finding suggests that complex systems can produce organizational misunderstandings that harm organizational adjustment: Organizations can misunderstand to adjust their structures according to architectural changes that have opposing trends at the upper and lower levels.

Keywords : complex system, architectural change, design evolution, integral and modular, architecture index

複雑システムの設計進化：
ソフトウェアのアーキテクチャ変化の測定

兵庫県立大学　立本博文

2010 年 12 月

要約：複雑なシステムでは、設計要素間の依存性は、モジュール内の依存性と、モジュール間の依存性の２つに区分することが出来る。モジュール間の依存関係が変化することをアーキテクチャ変化と呼ぶ。アーキテクチャ変化は産業構造や企業競争力に大きな影響を与えるため、多くの研究者が注目している。

設計進化プロセス中のアーキテクチャ変化を解き明かすため、多くのケース研究が行われている。しかしながらアーキテクチャ変化の概念や測定方法が整理されていないため実証的研究は少なく、アーキテクチャ変化の動的プロセスの詳細は分かっていない。

本研究では、モジュール間の依存性の強弱に基づいてアーキテクチャ指数を定義し、ソフトウェアアプリケーションのソースコードを用いてアーキテクチャを定量的に測定した。その結果、以下の３つの事が判明した。

第一に、どの階層でもアーキテクチャがインテグラルからモジュラーへと単調に変化するということはなかった。アーキテクチャは振り子のように揺り戻しを起こすことが分かった。インテグラルからモジュラーへとアーキテクチャが一直線に変化するという仮説は、いきすぎた単純化の可能性がある。

第二に、下位階層（コンポーネントレベル）では小さな変化が連続して発生する。それに対して、上位階層（アーキテクチャレベル）では大きな変化が時折生じる。このような階層間の変化パターンの差が「組織的認識限界」（小さな技術変化が、実はアーキテクチャ的には大きな変化であったという見落とし）を引き起こしている可能性がある。

第三に、上位階層と下位階層では、アーキテクチャ変化が逆に起こっている事が分かった。たとえば上位階層ではインテグラル化が進んでいるときに、下位階層ではモジュラー化が進んでいた。この事実は、組織が下位階層の変化に十分に対応した場合、上位階層では必要な対応と逆の対応を行っている可能性を示唆しており、「組織的認識限界」以上に深刻な「組織的錯誤」を引き起こす可能性を提示している。

キーワード：複雑なシステム、アーキテクチャルイノベーション、階層性、
インテグラルとモジュラー、設計進化

目次

1 節　アーキテクチャの動的プロセスの研究[1]

2000 年以降、アーキテクチャ研究では動的プロセスに研究者の注目が集まっている。動的プロセスとは、あるアーキテクチャ（元のアーキテクチャ）が様々な影響をうけて別のアーキテクチャへと変化するプロセスのことである（図 1)。アーキテクチャの動的プロセスには狭い意味の動的プロセスと広い意味の動的プロセスがあり、2 つをあわせて設計進化プロセスと呼ぶ。

狭い意味の動的プロセス研究は、「元のアーキテクチャ」がモジュラーオペレータ、分割基準の影響をうけて、「実現したアーキテクチャ」へと変化する過程を対象としている。それに対して、広い意味の動的プロセス研究は、狭い意味の動的プロセスに加えて、実現したアーキテクチャと分業構造の対応関係や、そこから生じるパフォーマンスとの関係も対象としている。アーキテクチャは、長期的には、分業構造と共進化していると考えられている。

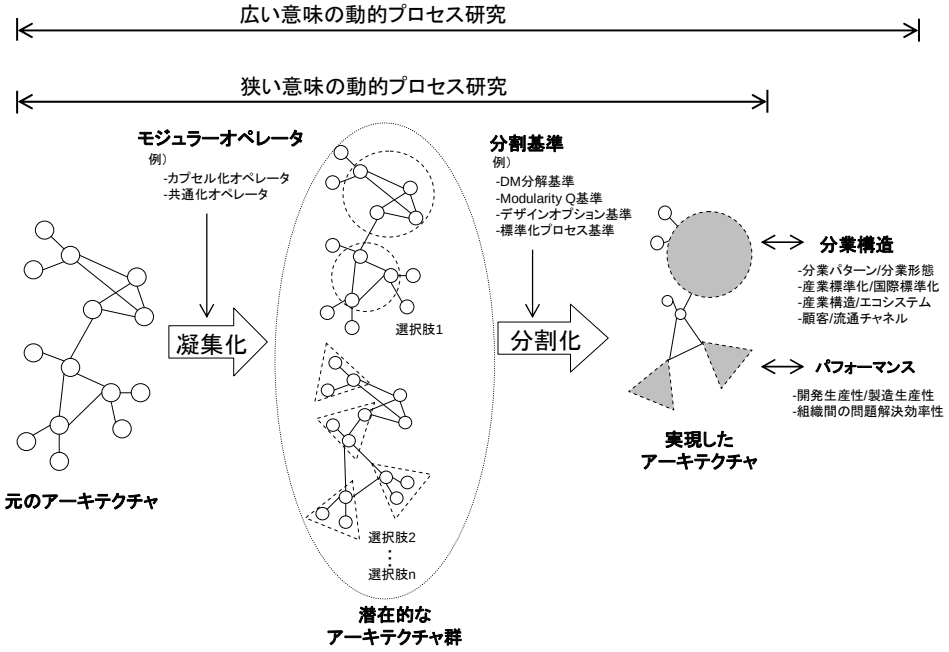


図 1 アーキテクチャの動的研究のフレームワーク

狭義・広義のどちらの動的プロセス研究でも、アーキテクチャがどのように変化するのか（アーキテクチャ変化）を定量的に測定することが動的プロセス研究の中心となる。

[1] 本論文は先の文献サーベイ（立本, 2010）に続くものである。より深い理解のために、本論文ともに同論文を読まれることを勧める。

しかしながら、実際に製品アーキテクチャを測定することは難しく、いままでアーキテクチャ変化を定量的に測定した研究は、いくつかの例外(MacCormack, Rusnak, Baldwin, 2006)を除いて、非常に少ない。また測定方法にコンセンサスが出来ているとも言い難い。

本稿では、まずアーキテクチャ変化をどのように測定可能にするのかを議論しながら測定方法を検討する。つぎに、その測定方法をソフトウェアアプリケーションに対して適用して、19 回のバージョンアップ中にどのようなアーキテクチャ変化が生じたのかを分析する。

2 節　設計進化の動的プロセスの測定方法

アーキテクチャというアイディアは、設計要素間の依存性（結合状態）をシステムの特徴として扱おうというアイディアである(Simon, 1962)。アーキテクチャの 2 つの状態（インテグラルとモジュラー）は分業構造に直接影響するため、この２つの状態間の変化に研究が集中している。

多くのケース研究では、インテグラルとモジュラーという２つの状態を基に、アーキテクチャの変化を分析しようとする枠組みが一般的である。しかし、未だこのフレームワークが十分に操作化されているとは言えない。①何を測定対象とするのか②インテグラルとモジュラーという「モジュール間の依存性」というアイディアをどう表現するかについて、定式化されていないのである。

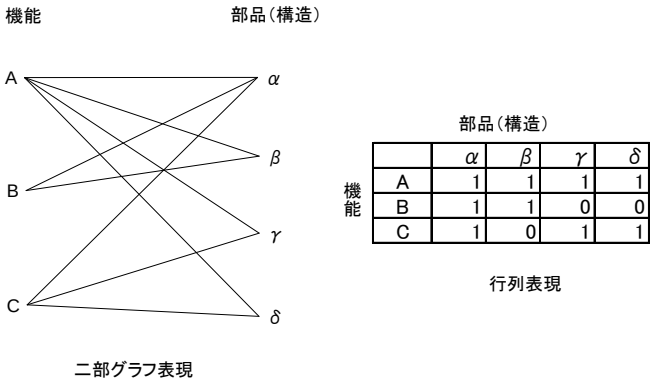
たとえば多くの研究では、インテグラル化と凝集化が混同されて議論されている。インテグラル化とはモジュール間の依存性が強化されることであり、凝集化とはモジュール内の依存性が強化されることである。インテグラル化と凝集化は似て非なるものであり、分業構造にあたえる影響も異なる。前者は組織間統合を促進するが、後者は組織間統合を促進するとはいえない。凝集化は、場合によっては、強力に組織の独立性を強化するかもしれない。インテグラル化と凝集化の錯誤は比較的早期から指摘されているが（青島・武石, 2000; 藤本, 2000）、多くの研究でこの２つを区別した操作化は行われていない。たとえばアーキテクチャの指標として単純に依存関係の密度を用いる事は、この錯誤を犯していると言える。

以下では①「測定対象」について 2.1 で、②「インテグラルとモジュラー」について 2..2 で説明する。

2.1 測定対象：ＤＳＭと構造行列

測定可能な方法で、アーキテクチャを定義したのは Ulrich(1995)である。彼は、測定対象としてシステムの機能と部品の対応関係をアーキテクチャとして測定するというアイディアである(Ulrich, 1995)。機能と部品をおのおのリストアップし、機能要素と構造要素の対応関係の二部グ

ラフができる(図 2 左)。この二部グラフ間の連結度合いを見ることによって、アーキテクチャを分析することが出来る。実際には、この 2 部グラフを行列化したものを測定対象にする（図 2 右）。



	α	β	γ	δ
A	1	1	1	1
B	1	1	0	0
C	1	0	1	1

図 2 グラフ表現と行列表現

「機能×構造」行列をアーキテクチャ測定の基本形と考えたときに、そこから２つの派生的な測定行列を考えることが出来る。一つは、「機能×機能」行列である。これは、「機能×構造」行列にたいして、その転置行列との積を計算することで、「機能×機能」「構造×構造」行列を作成することができる。「機能×機能」行列の各要素は、ある機能と別の機能の依存関係の数となる。「構造×構造」行列の場合、ある部品と別の部品との依存関係数となる

A＝「機能×構造」行列

$$= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

${}^{t}A$＝「構造×機能」行列(${}^{t}A$はAの転置行列)

$A \times {}^{t}A$＝「機能×機能」行列

$$= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 2 & 3 \\ 2 & 2 & 1 \\ 3 & 1 & 3 \end{pmatrix}$$

${}^{t}A \times A$＝「構造×構造」行列

$$= \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 1 & 2 \\ 3 & 3 & 1 & 2 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 1 & 2 \end{pmatrix}$$

図 3 「機能×構造」行列

「機能×機能」行列は、DSM(Design Structure Matrix)と呼ばれる。DSM の文脈では、機能は設計パラメタとよばれ、設計パラメタ間の依存関係が主な測定目標となる（Baldwin and Clark, 2000）。

「機能×機能」行列が測定対象になる一方で、「構造×構造」行列を測定目標にするアプローチも当然考えられる。「構造×構造」行列を構造行列と呼ぶ[2]。これら３つの行列は、結局は問題解決がどのように依存しているかを表している（図 3）。本論文では、構造行列を測定の目標としている。

2.2 インテグラルとモジュラーの表現について

インテグラル/モジュラーの行列表現

アーキテクチャの動的プロセス研究では、インテグラルな状態とモジュラーな状態の間でどのようにアーキテクチャが変化するのかを明らかにする事が大きな課題である（立本, 2010）。アーキテクチャがモジュラーアーキテクチャ、もしくはインテグラルアーキテクチャをとるとき、行列表現ではどのようになるのかを考えてみよう。

モジュラー型アーキテクチャの場合、言い換えれば、機能と構造が一対一関係にある時、「機能×構造」行列は、対角成分のみ≠0、その他の成分=0 となる。つまり、モジュラー化とは、DSM、構造行列が対角化されることと等しい。この場合、対角成分がモジュールである。

インテグラルアーキテクチャの場合は、モジュール間の依存性が高い状態であるので、対角成

[2]一般的なコンセンサスがあるわけではないが、本研究では以後「構造×構造」行列を構造行列と呼ぶことにする。

分間の値が非零になる。すなわち、対角成分とその他の成分≠0 となる。DSM、構造行列において同様のことがいえる。（図 4）

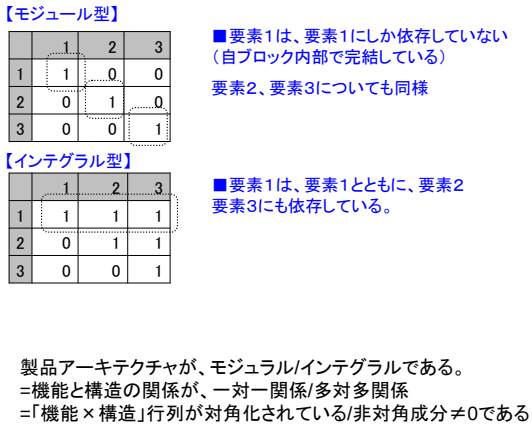


図 4 アーキテクチャの行列表現

縮約化（モジュール化）の重要性

アーキテクチャ変化の研究では、インテグラルとモジュラーという２つのアーキテクチャの状態に注目している。インテグラルとはモジュール間の結合が複雑に行われている状態であり、モジュラーとはモジュール間の結合が簡明に行われている状態である。アーキテクチャの２つの状態（インテグラルやモジュラー）は、分業構造に大きな影響を与えていることが知られている。

多くの研究では、インテグラルとモジュラーという二項対立によって分析が行われている。注意深く概念の操作化を行えば、このフレームワークは間違いではない。しかし、多くの研究では拙速な方法によって測定を行っているため、現象を正しく解釈することが難しい。一番多く行われる間違いは、「縮約化」というプロセスを無視する事である。縮約化はモジュール化とも階層化とも呼ばれる。

インテグラル化とは、モジュール間の結合状態が強まる事である。これと似て非なるものは凝集化で、モジュール内の結合状態が強まる事である。インテグラル化と凝集化は厳密に区別する必要がある。いいかえれば、モジュラー化（モジュール間の結合の簡明化）とモジュール化（モジュール内の結合の凝集化）を区別する必要があるのである。あるアーキテクチャがインテグラルなのかモジュラーであるのかは、モジュール化後に判明する。モジュール化（縮約化・階層化）が行われた後の、モジュール間の結合状態が問題なのである。

モジュラーオペレータ

どのような縮約化（モジュール化）がおこなわれるのかは、モジュラーオペレータによって決定される。モジュラーオペレータは複数存在する。たとえば Baldwin and Clark(2000)は、具体的な設計活動例から 6 つのモジュラーオペレータを概念化している

ここではより本質的であると思われる２つのオペレータを紹介する。立本(2010)は、オブジェクト指向設計方法から「カプセル化オペレータ」と「共通化オペレータ」の２つのモジュール化方法がモジュラーオペレータとして本質的であるとしている。

カプセル化オペレータは、設計要素間の依存性を隠蔽することによりモジュールを作ろうとするモジュラーオペレータである。カプセル化オペレータが作用すると、部品をいくつかあつめたサブアッセンブリータイプのモジュールが作られる（青木, 1995）。

共通化オペレータは、おなじような設計要素をまとめて共通化することによりモジュールを作ろうとするモジュラーオペレータである。共通化オペレータが作用すると、世の中のねじを全て共通ねじで統一しようというような標準化タイプのモジュールが作られる。

実際の設計進化プロセスでは、これら複数のモジュラーオペレータが同時に作用しており、モジュラーオペレータ毎に影響度合いが異なる。各モジュラーオペレータの影響度合いは、経験的な基準や統計的な手法から推定することになる。

アーキテクチャインデックス

モジュール間の結合関係の状態をしるためにアーキテクチャインデックスを定義する（図 5）。縮約後の行列（イメージ行列）に対して、モジュールが成立している行列の密度をアーキテクチャインデックスとする。ただし対角成分に関しては密度計算から除外する[3]。

アーキテクチャインデックスは、モジュール化（縮約化）後の密度であることに注意が必要である。そのためどのような粒度のモジュールを基底とするかで、アーキテクチャインデックスは変化する。小さいモジュール粒度を基底とすれば低階層のアーキテクチャを測定することができる。大きいモジュール粒度を基底とすれば高階層のアーキテクチャを測定することが出来る。

[3]本研究の後半でおこなったソフトウェアの測定では、全ての対角成分でモジュールが成立していた。今回の例では、約 1000×1000 行列を 4x4 行列まで縮約した結果、すべてのイメージ行列で対角成分≠0 であった。

図 5 アーキテクチャ指数

3 節　ソフトウェア製品への適用

ソフトウェア製品にどう適用するのか？

以上で今回のアーキテクチャ測定の基本的な戦略を述べた。これをソフトウェア製品に適用するためには、どうすればよいのだろうか。

ソフトウェア製品の設計図は、そのソースコードである。ソースコードには、いくつもの機能を実現するために大量の関数が定義されている。関数をソフトウェアの部品ともいえる。「構造×構造」行列を測定するために、ソフトウェアの部品である関数同士がどのような構造で定義されているかを分析すれば良い。関数同士がどのような呼ばれ方をしているのかを、コール構造とよび、グラフで示した物がコールグラフである。

ソースコードの構造を分析するということは、関数同士のコール構造の分析をすることと同じである。コール構造のグラフ表現をコールグラフと呼ぶ。このコールグラフから、構造行列を作成し、分析対象とする。

測定対象プロジェクト

測定対象が関数のコール関係を反映した構造行列を作成するために、ソースコードにアクセス

する必要がある。商用のソフトウェアではソースコードにアクセスすることは困難である。このため、オープンソースプロジェクトである apache プロジェクトを今回の測定の対象とした。オープンソース製品であれば、ソースコードの解析を行うことが出来るからである。

apache は、ホームページを公開するための WWW サーバ・ソフトウェアである。フリーソフトウェアでありながら高機能であるため、世界で最も利用されているWWWサーバである。

Apache [4]は、1997 年以降 www サーバでトップシェアを誇っており、2005 年の統計では、２位のマイクロソフト社の WWW サーバ製品(IIS)の 20.54%に対して、Apache は、69.37%のシェアを獲得している。市場に受け入れられた製品である。Apache は、いくつかの系列に分かれて開発されており、今回の分析対象は 1.3 系列とした。

CONCOR による縮約化

現実の設計進化プロセスでは、複数のモジュラーオペレータが同時に作用しているため、どのモジュラーオペレータがもっとも影響しているのか等を考慮する必要がある。しかしながら、今回の測定は予備的な研究であるので、効果がわかりやすいようにモジュラーオペレータを１つだけ適用し、共通化オペレータの効果を測定する。

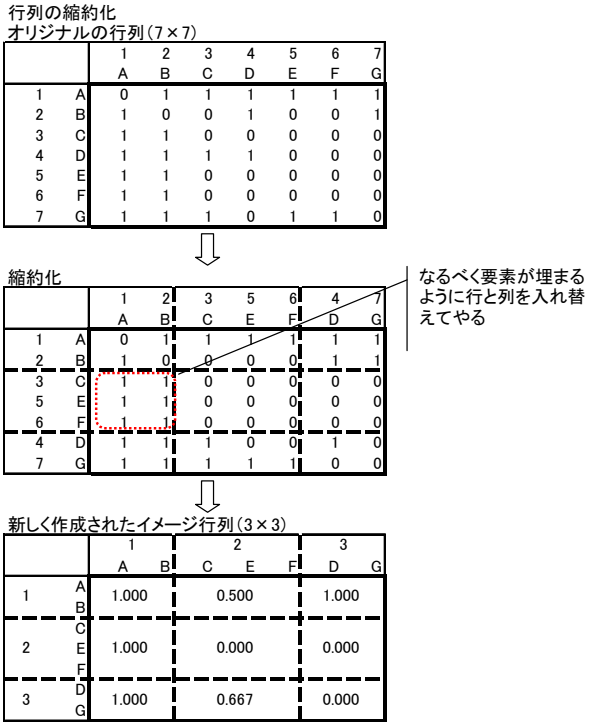


図 5　CONCOR を用いた縮約化の例

[4]Netcraft 社調査発表(http://news.netcraft.com/archives/web_server_survey.html, 2004 年 6 月 19 日アクセス)

今回の測定では「共通化オペレータ」を表現するため、CONCOR アルゴリズム(Schwartz, 1977)を用いた。CONCOR アルゴリズムは、ピアソンの積率相関係数を用いて似たような関係性を持つ設計要素をまとめていく方法である。これにより同じような関係構造をもっているモジュールは共通化されていくというプロセスを表現することが出来る。図 5 では、7×7 行列を 3×3 行列に縮約した例を示している。縮約化した行列のことをイメージ行列と呼ぶ。

測定手順まとめ

測定手順をまとめると、次のようになる。

①ソースコードより、関数の呼び合い構造（コール構造）を抽出する。

②関数の呼び合い構造から構造行列を作成する。

③構造行列に CONCOR アルゴリズムを用いて縮約化しイメージ行列を作成する。

④作成したイメージ行列のアーキテクチャを分析する。

4 節　適用結果・考察

１　縮約化の結果

apache 1.3 系列にはいくつかのリリース・バージョンがある。1.3 系列ではもっとも初期のリリースであるリリース 1.3.0 を用いて、縮約化を行った。関数同士の構造行列の大きさは 984×984 行列であった。これを 4×4 行列に縮約[5]した。その結果が図 6 である。

[5] 縮約には UCINET 6(Borgatti et al., 2002)の CONCOR プロシージャーを用いた。プロシージャーのパラメタはすべてデフォルト値を使用した。

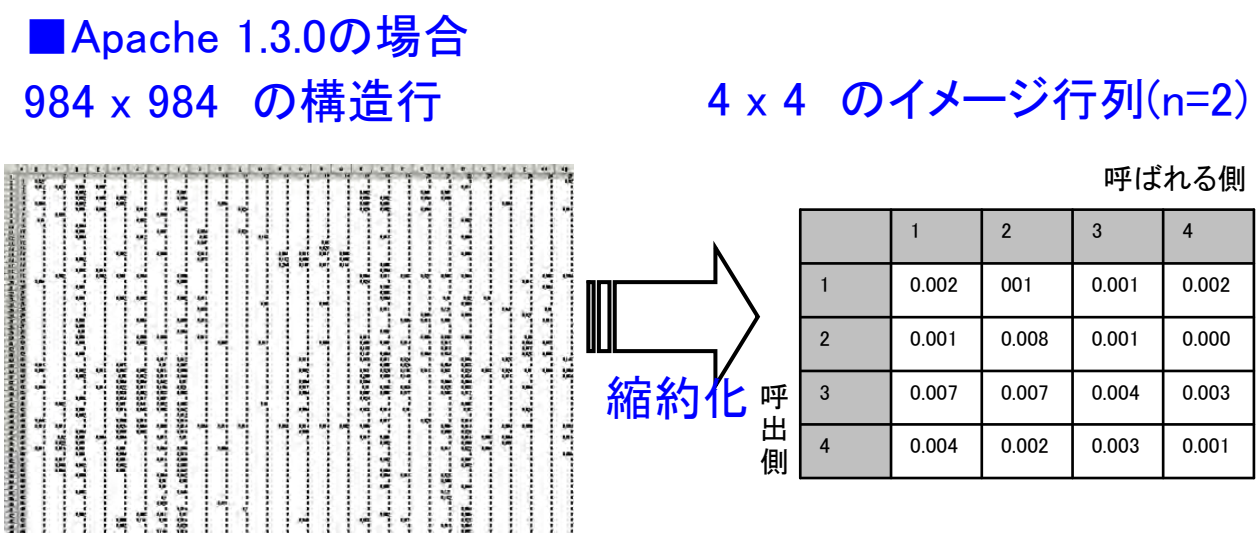


図 6 縮約化とイメージ行列

この縮約化が基本的な設計思想のリファレンスとして有効であるかを確認するため、4×4 のイメージ行列の行要素の関係を見てみよう。確認する際に各行に名称づけをした。名称づけに際して、各行要素に含まれる関数名とその関数が属するファイル名およびソースコード内のコメントを参考にした。その結果が表 1 である。

行	77jn 名称	内容
1	プロトコル解析関数	http プロトコルの解析およびそれに関係のある文字列操作
2	プロセス制御エンジン	http リクエストのサイクルの制御(プロセス制御)およびその記録
3	リクエスト処理エンジン	http リクエストと関係する各処理モジュール(コンテンツ種類の判別や権限制御)の調整処理
4	共通関数	設定ファイルやサーバの状態から変数を設定する

表 1 各行要素と内容

イメージ行列を確認すると、4 つの点で特徴があることがわかる。①(3,1)の値が 0.007 と高い。② (3,2)の値が 0.008 と高い。③(3,2)の値が 0.007 と高い。④また、(4,4)=0.001 の行列要素の値が小さいこともわかる。この 4 つの特徴が、従来の apache のアーキテクチャの説明と整合的であるか確かめておこう。

従来 apache のアーキテクチャの説明に使われている構成図との比較をした。名前を付けた要

素がどこに位置づけられるのかの対応を示した物が図 7 である。

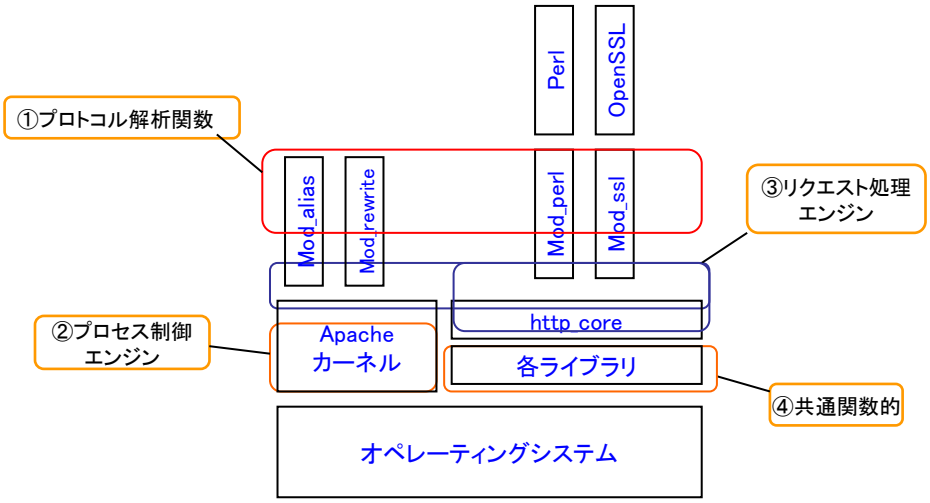


図 7 Apache の構成図

1 点目に、①部分のプロセス制御エンジン内の関係性が強い。プロセス制御エンジンは、http リクエストの処理サイクルを制御する物で、最も www サーバの処理効率の関係する部分である。パフォーマンスを高めているという意味で、この部分が内部依存度を高めていると解釈できる。

2 点目に、②部分で、リクエスト処理エンジンとプロセス制御エンジンの関係性が高い。プロセス制御エンジン内では、コンテンツ毎にいろいろな処理を行う必要性があり、この部分をリクエスト処理エンジンが行っている。そのため、この部分の値が高いことは整合的である。

3 点目に、③部分で、プロトコル解析関数とリクエスト処理エンジンの関係性が強い。これは、http リクエストを処理するのに、各プロトコル解析をする必要があるためで、この部分の関係が強いことは整合的である。

4 点目に、④の共通関数部分は、自分自身への依存度が高くない((4,4)=0.001)。共通関数という性質のため、他のモジュールから呼ばれたり、逆に呼んだりすることの方が多く、共通関数内部での呼び合いは、少ないことを意味しており、これも整合的である。

縮約化によって、要素間の大まかな関係を維持したまま構造行列がサイズの小さな行列(=イメ

ージ行列)になったことがわかる。構造行列がわかっているときに、それを縮約化してやることで、アーキテクチャ測定に利用できるということが確認できた。

5 節 動的プロセス分析

5.1 時系列分析

バージョン変遷

前項では、構造行列の縮約化という手法を用いて、apache の構造行列のイメージ行列を作成し、アーキテクチャの観察をおこなった。これを用いて、アーキテクチャの時系列分析を行ってみたい。アーキテクチャは、1 時点の状態が問題なのではなく、どのように変化するかという点が、実務的な点からも理論的な面からも重要である。

前出の Apache 1.3 系列には、いくつかバージョンがあると紹介したが、分析の時点では、19 バージョンの情報を収集することが出来た。各バージョンのリリース時期をまとめると図 8 になる。

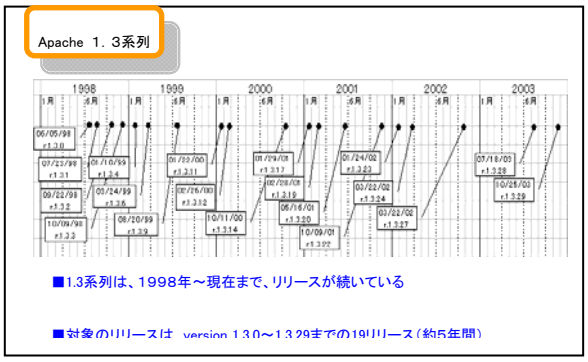


図 8　Apache のリリーススケジュール

各リリースの期間を見ると、1998～1999 年にかけては頻繁なリリースを繰り返し、2000 年以降は、ある程度機能が成熟化したために、リリースの期間が長くなっている。1.3 系列は、2003 年以降もリリースされているがリリースの頻度は少なくなっている。ソフトウェアの規模を見てみると、最初のリリースでは約 39,000 行だが、19 回のリリースの後には、約 51,000 行となっている。ただし、この間は、約 5 年間程度あり、むしろサイズの増大はゆるやかであったと思われる。

アーキテクチャ指数

バージョン間のアーキテクチャの変化をアーキテクチャという視点から比較するために、アーキテクチャ指数を定義する(図 9)。定義自体は簡単であり、モジュールが成立している行列成分の密度をとっている。アルゴリズムによる縮約の場合、ケースによっては対角成分≠0 にならず、モジュールが成立しないことがある。その場合を防ぐために、対角成分≠0 であるという条件をおいている。しかしながら、今回の例では、約 1000×1000 行列を 4x4 行列まで縮約した結果、すべてのイメージ行列で対角成分≠0 であった。

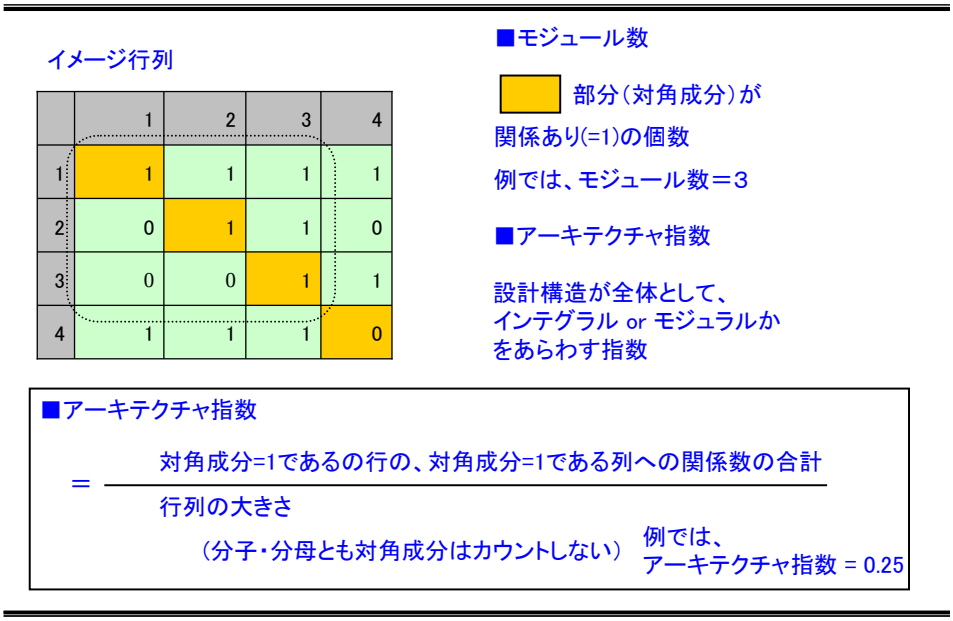


図 9 アーキテクチャ指数

19 回のバージョン間で構造行列のサイズを見てみると、最初の構造行列のサイズは、984×984 行列であり、19 回のリリース後でも 1122×1122 行列程度であった。サイズ的な違いはそれほど大きくない。各リリース・バージョンの構造行列を 4×4 のイメージ行列に縮約してやり、上記で定義したアーキテクチャ指数をプロットしたものが図 10 である。

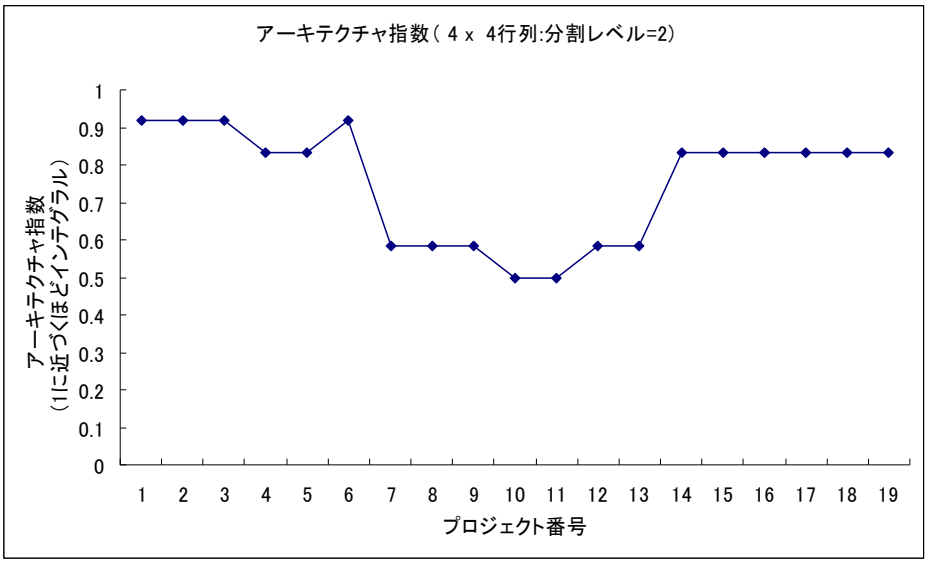


図 10 アーキテクチャ指数の推移

バージョン毎のアーキテクチャ指数を観察すると、リリース 6 から 7 の段階で大きくモジュール型に落ち込み、その後リリース 13 まで同じような値で推移する。リリース 13 からリリース 14 にかけて、インテグラル型にふれ、現在に至っている。ちょうど、前半部分はインテグラル型からモジュール型にアーキテクチャ指数がふれ、その後モジュール型からインテグラル型に移行している。

この理由を知るために、リリースに付属している開発ログ[6]を確かめると、リリース 7 の時点で、システムに新しいモジュールを追加した旨が記載されている。新モジュール追加により、apache のシステム全体のモジュール度が上がったのだと思われる。一方、リリース 14 の開発ログを見てみると、セキュリティーの問題が高まったため、変更を行った旨が記載されている。おそらくセキュリティー問題は、その性質上、一つの部分で解決できる様な物でなく、該当する部分すべてに変更が必要であり、その結果、コードの見直しの際に、関数の呼び合い構造も変化したのではないかと思われる[7]。

5.2 階層分析

階層ごとに動的プロセスの推移を検討したものが図 11 である。各階層とは分割レベルが大きいものを第 1 階層とし、分割レベルを 2 倍細かくする従って第 2 階層、第 3 階層というようにし

6 すべての apache のリリースには、CHANGES という名称の開発ログファイルが添付されている。

た。第 1 階層では、全体システムを 4 つのモジュールで説明しようとしているが、第 2 階層では 8 つのモジュールで説明している。最終的に第 6 階層（全体を 128 のモジュールで説明する）まで計算を試みた。

この階層分析では、階層が深くなればなるほど、細かい設計要素レベルの動的プロセスを反映するようになる。すなわち、深い階層（第 5 階層や第 6 階層）はコンポーネントレベルの動的プロセスを示している。反対に浅い階層（第 1 階層や第 2 階層）では、全体を大きなモジュールで捉えたときの動的プロセスをしめしており、アーキテクチャレベルの動的プロセスを示していると言える。

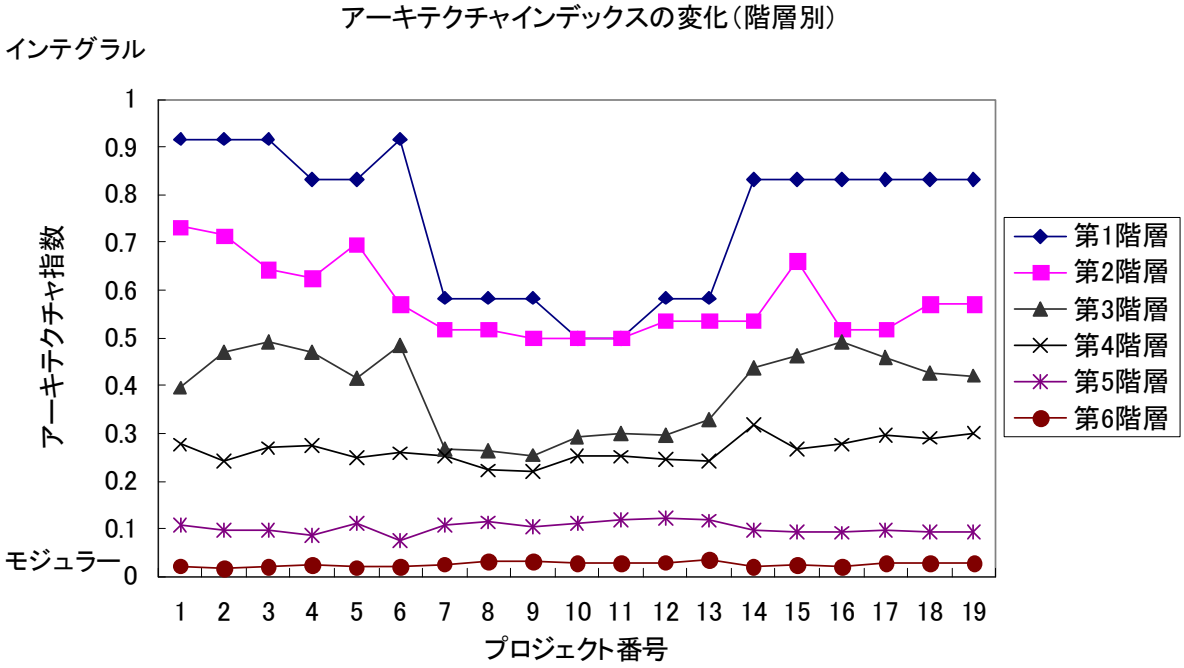


図 11　階層毎のアーキテクチャ指数の変化

図 11 から次の 3 つの事が分かる。

i)　アーキテクチャ変化の推移について

全ての階層においてインテグラルからもモジュラーへと単調にアーキテクチャが変化したケースは存在しない。ほとんどの階層において、一時はインテグラルからモジュラーへとアーキテクチャがシフトしたものの、最終的には、もう一度モジュラーからインテグラルへとアーキテク

チャがシフトしているケースが見られる。とくに第1階層、第3階層ではその傾向が顕著である。

ii) 階層毎のアーキテクチャ変化の強弱について

階層毎にアーキテクチャ変化の幅が異なる事が観察された。より下位の階層（第 5 階層や第 6 階層）はアーキテクチャ指数の上下の振れが小さいのに対して、より上位の階層（第 1 階層や第 2 階層）はアーキテクチャ指数が大きく上下に振れる。これは、より小さいモジュールの粒度でシステムを見たときにはあまり変化が無い場合でも、より大きなモジュールの粒度で見た場合にはシステム的な変化が起きていることを意味している。

すなわち下位階層（コンポーネントレベル）では小さな変化しか発生していなかったとしても、それは集合的には上位階層（アーキテクチャレベル）では大きな変化が生まれている事を示している。この事実は、Henderson and Clark(1992)の「コンポーネントレベルでの変化に適応した組織構造をとると、アーキテクチャレベルの変化を見逃す」という主張を裏付けるものであり、今回の発見である。

iii)階層間のアーキテクチャ変化のパターン差について

	第1階層	第2階層	第3階層	第4階層	第5階層	第6階層
第1階層	1.000					
第2階層	0.698**	1.000				
第3階層	0.915**	0.534*	1.000			
第4階層	0.565*	0.086	0.614*	1.000		
第5階層	−0.737**	−0.224	−0.767**	−0.478*	1.000	
第6階層	−0.750**	−0.644**	−0.724**	−0.316	0.498**	1.000

**1%有意 *5%有意

表 2 各階層間の相関係数

階層毎のアーキテクチャ変化に関してより興味深い事実がわかった。「上位階層のアーキテクチャ変化」と「下位階層のアーキテクチャ変化」が逆相関しているのである。表 2 は 19 回のバージョンアップ期間中に、第 1 階層から第 6 階層までのアーキテクチャ指数がどのように相関しているのかを示している。例えば第 1 階層と第 3 階層は強い相関(0.915)を示しており、アーキテクチャ指数としては同じような動きをしていたことが分かる。もしも第 1 階層がインテグラルに振れたならば、第 3 階層もインテグラルに振れる確率が高いわけである。このように各階層間の相関を見ていくと、「第 1～4 階層」と、「第 5～6 階層」の間の相関係数が負になっている事に気がつく。「第 1～4 階層」と「第 5～6 階層」の間には大きなギャップがあるのである。

「第 1〜4 階層」をアーキテクチャレベル、第 5〜6 階層をコンポーネントレベルとしたとすると、アーキテクチャレベルとコンポーネントレベルでは、インテグラル/モジュラーの振れが逆に動いていたことになる。たとえばコンポーネントレベルでインテグラル化が進んでいたときに、アーキテクチャレベルではモジュラー化が進んでいたと解釈できるわけである。コンポーネントレベルの変化は単に小さいだけではなく、アーキテクチャレベルの変化とは逆に作用していたことになる。

「下位階層（コンポーネントレベル）の動的プロセスが、上位階層（アーキテクチャレベル）の動的プロセスとは、アーキテクチャ的に逆の動きをする」という事実発見は驚くべきものである。もしも下位階層の変化に的確に対応した組織構造をとったとすると、それは、上位階層の変化に対して必要な対応と逆の対応をとっている可能性があるからである。

6 節 インプリケーションと課題

今回の研究のまとめと考察

今回の研究では、構造行列の縮約化という手法を用いて、アーキテクチャの動的プロセスの測定を行った。構造行列を縮約化したイメージ行列を用いて、従来概念的に説明されていた「アーキテクチャ」と比較して、実証的に動的プロセスを把握した。測定の評価軸としては、インテグラル/モジュラーの軸を念頭に、アーキテクチャ指数を定義した。このアーキテクチャ指数を用いて時系分析と階層分析を行った。

本研究の第 1 の意義としては、アーキテクチャを定量的にとらえることが出来たという点を挙げることが出来るだろう。従来研究は

検証された動的プロセスは、従来の仮定とは異なるものであった。時系列分析からは、インテグラルアーキテクチャであったものがモジュラーアーキテクチャに変化し、さらに再びインテグラルアーキテクチャに変化するする様子が観察された。この事実から考察すると、複雑性軽減のためアーキテクチャは一直線にインテグラルからモジュラーへと変化するという仮定は、過度の単純化の可能性がある。

第 2 の意義として階層構造に関する分析を実証的に行ったことである。階層分析からコンポーネントレベルではそれほど大きな変化が無かったとしても、アーキテクチャレベルでは大きな変化が起こる場合がある事が確認された。この事実は、「コンポーネントレベルの変化に組織が十分適応すると、アーキテクチャレベルの変化を組織的に見逃す可能性が高まる」という従来の主張(Henderson and Clark, 1992; Christensen, 1992a, 1992b)を強く支持するものであった。既存研究は人工物そのものの構造データを用いた実証分析ではなかったが、本研究ではソフトウェアの構造データを用いながら実証的に仮説を支持する結果を得た。

第 3 の意義は、上位階層と下位階層では設計進化の方向が逆になる可能性がある事を明らかにした点である。この点は従来指摘されたことないものであり、大変興味深いものである。

コンポーネントレベルでインテグラル化が進んでいるときにアーキテクチャレベルではモジュラー化が進行し、逆にコンポーネントレベルでモジュラー化が進んでいるときにアーキテクチャレベルではインテグラル化が進んでいたことが、階層分析からわかった（図 12)。

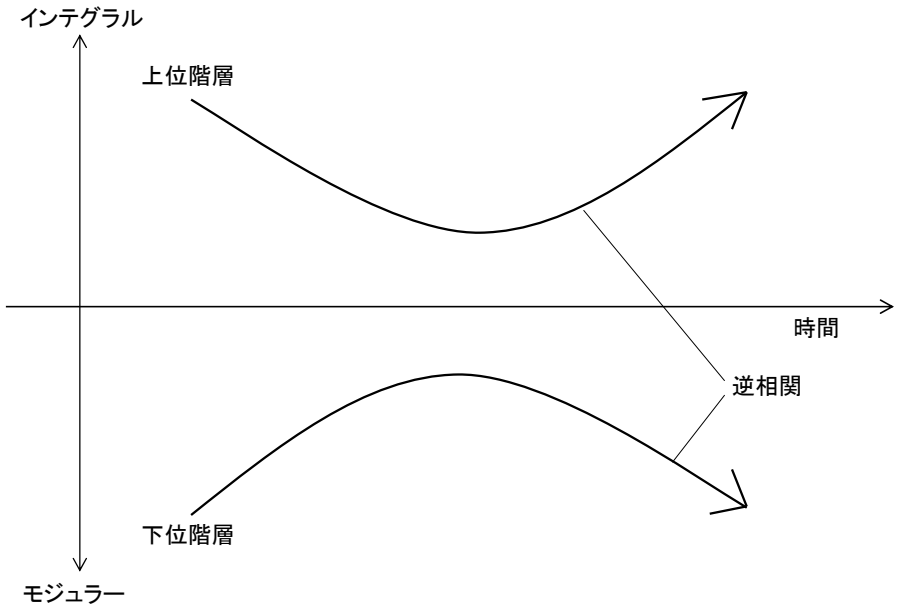


図 12 階層別のアーキテクチャ変化

この事実は組織マネジメント的に大きな洞察をもたらす。コンポーネントレベルのアーキテクチャ変化に組織が十分対応すると、アーキテクチャレベルの変化を「間違った方向に向かっている」と錯誤する可能性があるからである。

従来のアーキテクチャレベルのイノベーションに対する見解は「コンポーネントレベルでは小さな変化しか引き起こさない技術革新が、アーキテクチャレベルでは大きな変化が生じる可能性がある。だから、この小さな変化にさえも気がつけるような組織能力を持つべきだ」というものであった(Henderson and Clark, 1990)。しかし今回の階層分析で示されたのは、コンポーネントレベルとアーキテクチャレベルでは、設計進化の方向変化が逆という事実であった。この場合、コンポーネントレベルの小さな変化を見逃さないようにすればするほど、アーキテクチャレベルの変化の方向性を間違えるのである。

つまり今回の分析からの示唆としては「コンポーネントレベルとアーキテクチャレベルでは別々の組織的な認知能力が必要である」という点が挙げられる。コンポーネントレベルの観察を積み重ねたからと言って、それがアーキテクチャレベルの見解にならない。むしろコンポーネントレベルの観察はアーキテクチャレベルに錯誤をあたえる。だからアーキテクチャレベルの変化を分析する組織能力を、コンポーネントレベルの組織能力とは別に持たなくてはいけないのである。

最後に本研究の限界と今後の課題について述べる。本研究は特定のソフトウェアプロジェクトのデータをもとにしたものであり、さらなる一般化が必要な事は言うまでもない。データから導き出された結果は驚くべきものであったが、未だ明らかにされていない部分も多い。

本研究では、動的プロセスの観点からアーキテクチャ的に変化があることを明らかにしたが、どのような時にアーキテクチャ変化が起こるのかについては依然明らかになっていない。またコンポーネントレベルとアーキテクチャレベルの設計進化が逆相関するケースがある事も明らかになったが、どのような条件でこのようなことが起こるのかについては曖昧なままである。さらに、複雑な人工物のアーキテクチャ変化（動的プロセス）が、組織デザインやサプライヤーとの関係にどのような影響を与えるのか、パフォーマンスにどのような影響を与えるのかについては今回の研究では扱うことが出来なかった。これらの問題は今後の研究の課題であると思われる。

参考文献

Alexander, C. (1964) *Notes on the Synthesis of Form*, Harvard University Press.

Asanuma, B. (1989) Manufacturer-supplier Relationships in Japan and the Concept of

Relation-specific Skill, *Journal of the Japanese and International Economies*, Vol.3, No.1, pp.1-30.

Burt, R.S. (1976) Positions in networks, *Social Forces*, Vol. 55, pp.91-122

Baldwin, K. Y. and Clark, K. B. (2000) *Design rules: The power of modularity*, MIT Press. （邦訳、カーリス・Y・ボールドウイン、キム・B・クラーク（2004）『デザイン・ルール』安藤晴彦訳、東洋経済新報社.）

Borgatti, S.P., Everett, M.,G.. and Freeman, L., C. (2002) *Ucinet 6 for Windows: Software for Social Network Analysis*, Analytic Technologies.

Henderson, R.,M. and Clark, K., B. (1990) Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms, *Administrative Science Quarterly*, Vol.35, pp.9-30.

Clark, K. B. (1985) The interaction of design hierarchies and market concepts in the technological evolution, *Research Policy*, Vol. 14, pp.235-251.

Clark, K., B. (1989) Project scope and project performance: The effect of parts strategy and supplier involvement on product development, *Management Science*, Vol.35, No.10, pp.1247-1263.

Christensen, C., M. (1992a) Exploring the Limits of the Technology S-Curve, Part I: component Technologies, *Production and Operations Management*, Vol.1, pp.334-357.

Christensen, C., M. (1992b) Exploring the Limits of the Technology S-curve, Part 2: Architectural Technologies, *Production and Operations Management*, Vol. 1, pp.358-66.

Christensen, C. M. (1997) *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*, Harvard Business Press.

Chesbrough, H., W. and Teece, D., J.(1996) Organizing for Innovation: When Is Virtual Virtuous?, *Harvard Business Review*, 1996 January-February, pp.65-73.

Dyer, J. H. and Nobeoka, K.(2000) Creating and managing a high-performance knowledge-sharing network: the Toyota case, *Strategic. Management Journal*, Vol.21, pp.345-367.

Economides, N. (1996) The Economics of Networks, *International Journal of Industrial Organization*, Vol.14, pp.673-699.

Langlois, R., N. and Robertson, P., L. (1992) Networks and innovation in a modular system: Lessons from the microcomputer and stereo component industries, *Research Policy*, Vol. 21, pp. 297-313.

MacCormack, A., Runsnak, J. and Baldwin, C., Y. (2006) Exploring the structure of complex software designs: An empirical study of open software and proprietary code, *Management Science*, Vol.52, No.7, pp.1015-1030.

Newman, M., E., J. and Girvan, M. (2004) Finding and evaluating community structure in networks, *Physical Review E*, Vol.69, No.12, pp.1-16.

Robertson, P., L. and Langlois, R., N. (1995) Innovation, network and vertical integration, *Research Policy*, Vol.24, pp.543-562.

Schwartz, J.E. (1977) An Examination of Concor and Related Methods for Blocking Sociometric Data, *Sociological Methodology*, Vol.8, pp.255-282.

Shapiro, C. and Varian, H. R. (1999) *Information Rules*, Harvard Business School Press.

Simon, H., A. (1962) The Architecture of Complexity, *Proceedings of the American Philosophical Society*, Vol. 106, No. 6. (Dec. 12, 1962), pp.467-482.

Takeishi, A. (2002) Knowledge Partitioning in the Interfirm Division of Labor: The Case of Automotive Product Development, *Organization Science*, Vol. 13, No.3, pp.321-338.

Teece, D., J. (1986) Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy, *Research Policy*, Vol.15, pp.285-305.

Ulrich, K., T.（1995）Product Architecture in the manufacturing Firm, *Research Policy*, Vol. 24, pp.419-440.

Wiener, N. (1948) Cybernetics, MIT Press. (邦訳　『サイバネティクス第二版』(1961) 岩波書店)

Williamson, E. O. (1979) Transaction-Cost Economies: The Governance of Contractual Relations, *The Journal of Law and Economics*, Vol.22, pp.548-577.

青木昌彦(1995)『経済システムの進化と多元性：比較制度分析序説』東洋経済出版社.

青島矢一・武石彰(2001) 「アーキテクチャという考え方」所収 藤本・武石・青島(2001) 第 2 章.

小川紘一(2009) 『国際標準化と事業戦略―日本型イノベーションとしての標準化ビジネスモデル』白桃書房.

金光淳(2003) 『社会ネットワーク分析の基礎：社会的関係資本論にむけて』勁草書房.

立本博文(2010) 「設計進化のダイナミクス：複雑システムのアーキテクチャ研究の流れ」MMRC Discussion Paper, No.332, 東京大学ものづくり経営研究センター.

立本博文・小川紘一・新宅純二郎(2010) 「オープンイノベーションとプラットフォームビジネス」『研究技術計画』Vol. 25, No. 1.(forthcoming)

立本博文・高梨千賀子(forthcoming)「標準規格をめぐる競争戦略 ーコンセンサス標準の確立と利益獲得を目指してー」 所収 新宅純二郎編『国際標準と新興国市場』白桃書房.

徳田昭雄(2008)『自動車のエレクトロニクス化と標準化―転換期に立つ電子制御システム市場』晃洋書房.

徳田昭雄・立本博文・小川紘一編(forthcoming)『自動車組込みシステムの開発と標準化：欧州オープン・イノベーションの実態』晃洋書房.

藤本隆宏(2001)「アーキテクチャの産業論」所収 藤本・武石・青島(2001) 第 1 章.

藤本隆宏・武石彰・青島矢一編（2001）『ビジネス・アーキテクチャ』有斐閣.

宮田由紀夫(2001) 『アメリカの産業政策－論争と実践』八千代出版.

室田一雄(2004) 混合行列の正準形と階層構造『数学セミナー』Vol. 513, pp.38-43.

安田雪(2001) 『実践ネットワーク分析 : 関係を解く理論と技法』新曜社.

鈴木努(2009) 『ネットワーク分析』共立出版.