オブジェクト指向によるシステムの進化メトリクスの検討

中谷 多哉子[1]　友枝 敦[2]　酒匂 寛[2]　玉井 哲雄[1]

概要

オブジェクト指向システムの生産性や品質を計測するために，メトリクスの研究が多く行われてきた．しかし，漸進的な開発によって進化するオブジェクトの構造を，定量的に捉える研究はほとんど行われていない．構造的な進化に対処する研究としては，データベーススキーマの進化に関するものが発表されている．

我々は，進化を定量的に計測することで，オブジェクト構造の成長プロセスを明らかにし，明らかになった成長プロセスに基づいた開発計画の立案を提案したいと考えている．

本研究では、オブジェクトの進化を明らかにする進化メトリクスについて検討し、実際のアプリケーション開発事例に適用してデータを収集し，その視覚化を試みた。その結果、進化メトリクスに基づいた時系列の計測を行うことによって，システムの進化と計測結果との関係，進化の軌跡，および進化によっても維持されるクラスの定量的な特徴を明らかにできることがわかった．

1. はじめに

最初に開発されたソフトウェアが拡張され，他のソフトウェアへ移行していく過程において，生物学における進化論と対比させると，いくつかの類似性が発見できる．

進化論では，常に環境に適応した優勢な種が，有利な位置を占めて子孫を残し，それが種の進化をもたらすと考えられている[5] [14] ．ソフトウェアの種の分類には，機能や適用される問題領域，規模，アーキテクチャなどの手がかりがある．ここで，同種のソフトウェアが複数存在する場合は，商業的に，あるいは利用者にとって有利なソフトウェアが市場で生き残っていく．これは生物の自然選択に対応すると考えることができる．また，生き残ったソフトウェアから派生して開発されたソフトウェアは，生物の種の進化と対応する．

とくにオブジェクト指向で開発されるソフトウェアでは，再利用という手段によって同じクラスライブラリやデザインパターン[7] が複数のソフトウェアで利用されることが多い．再び進化論を用いてこれを類推すると，再利用されるクラスライブラリやデザインパターンは遺伝子と言うことになるのかも知れない．

ところで，ひとつのソフトウェアに注目した場合，保守に該当する変化は，生物に例えると何になるのだろうか．単純に考えると，個々のソフトウェアはひとつの個体なので成長という言葉を当てはめたくなる．しかし，生物の成長は個体が生まれたとき，すでにプログラムされている変化を辿るのに対して，ソフトウェアの場合は，その成長を開発時に定義できることは少ない．したがって，ひとつのソフトウェアの変化に対して成長という言葉を使うことは，必ずしも適切とは思えない．ソフトウェアは，多くの変化する環境，たとえば，利用者要求の変化や，コンピュータ環境の変化，ライブラリの変化，方法論の変化などに随時適応していくものである．この特徴は生物の成長というよりも，むしろ進化の仕方に似ている．ただし，生物学は種の進化を論じるが，個々のソフトウェアの場合は個体の進化について論じることになる．

そこで，ひとつのソフトウェアにおける運用／保守による構成要素の変化や，構成要素間の関係の変化を進化と呼ぶことにした．

もちろん，ソフトウェアの進化と生物の進化には違いがある．生物は自然に，環境に適応していく方向に進化すると言われているが，ソフトウェアの進化の場合は設計者によって人為的に進められるのが

1　東京大学大学院総合文化研究科広域科学専攻 広域システム科学系
2　株式会社SRA　SIビジネス第二部オブジェクト指向グループ

普通である．この違いは，われわれにとって好都合である．なぜならば，われわれにはソフトウェアの進化を適当な設計方法論で制御できる可能性があるからである．

従来からソフトウェアの開発では，保守による構造の劣化が問題になり，保守コストの増大も大きな課題として取り上げられ，研究されてきた[3] [12]．開発しているソフトウェアをいつまで拡張し，どの時期に寿命を設定し作り替えるのか[22]，どの部分を次のソフトウェアに遺伝子として伝えていけばよいのかなど，開発計画ではいくつかの戦略を練る必要がある[10] [21]．これは，従来のシステム開発のみならず，オブジェクト指向で開発されたシステムでも変わりはないだろう．

現在のところ，いくつかの実用化されているオブジェクト指向開発方法論は，新規開発やリバースエンジニアリングによる再開発を主な対象にしており[20]，オブジェクト指向システムに対して，運用／保守／移行までの開発計画を対象にした研究はほとんど行われてこなかった．ソフトウェアの進化の過程を研究することは，運用／保守および移行を支援するために効果があるだろう．また，類似ソフトウェアの進化の傾向がわかれば，進化を制御しながら移行の時期を決定することも可能になるだろう．

そこで，オブジェクト指向システムの進化を視覚的に捉える試みを行った．まず最初に，システムの進化の様子を客観的に捉えるために，進化の定量尺度として進化メトリクスを検討した．そして，この進化メトリクスを実際のオブジェクト指向システムに適用し，進化の計測実験を行い，メトリクスとしての妥当性の評価を行い，その有効性を導いた．

本論文の構成は以下のようになっている．

2.で，進化を観察する対象と，各対象の進化メトリクスを検討する．3.で進化メトリクスを適用した計測実験の結果を示す．4.では，計測実験の結果をもとにメトリクスを評価し，それぞれのメトリクスから得られた知見を示す．その後で，保守や再構築を含めた開発計画の指針を示す可能性について議論する．5.では，関連研究と今後の研究について紹介する．

2. 進化メトリクスの検討

2.1. 進化を観察する対象

オブジェクトの特性を表す要素には，オブジェクトの属性，操作，および継承構造の他に，インスタンス間の関係である静的な集約構造や動的な関連が存在する．また，クラス，属性，および操作の名称も重要な要素である．とくに操作の名称は，多相性という点において，その引数の名前や型とともに注意が払われる．

システムの進化では，以上の要素がさまざまな原因によって変化すると考えられる．われわれは，このような変化を観察するにあたり，次の4つを観測対象として設定した．すなわち，システム，クラス，メッセージ，メソッドである．これらの間には包含関係がある．たとえば，システムでは全体の変化を観察でき，クラス，メッセージおよびメソッドでは，システムの進化を実現している個々の対象の変化を観察できる．

また，進化を捉える手段には定量的な観測と定性的な調査がある．さらに，各対象に対して適用できる進化メトリクスは，静的メトリクスと動的メトリクスに分けることができる．静的メトリクスはプログラムの構造を定量的に観測するためのメトリクスであり，動的メトリクスはプログラム実行中に計測できる呼び出し関係や動的な関連などを観測するためのメトリクスである．

本論文では，とくに，定量的な観測を行うための進化メトリクスのうち，静的なメトリクスについて検討する．

2.2. 進化メトリクスの検討

オブジェクト指向メトリクスとして，Chidamberらによって提案されている6つのメトリクス（以下CKメトリクス）は，システムやクラスの進化を捉えるメトリクスとして提案されたものではない．しかし，個々の版におけるクラスの複雑度を捉えるためには有効である．進化を捉える場合は，これらのメトリクスを時系列で計測する必要がある．また，その他にも，クラス構造の変化を明らかにできるようなメトリクスが必要である．

以下に各観測対象に適用する静的な進化メトリクスと，その意味について説明する．ここで，定量的な観測では，進化による変化は次の式によって求められる結果を得ることができる．

$$E_n = E_{n-1} + \Delta E_n$$
$$\Delta E_n = Add_n - Sub_n$$

ここで，ΔE_n: 進化によって計測された変化量，E_n :ある時nの観測値，Add_n: 進化によって追加された量，Sub_n: 進化によって削除された量，である．Add_nおよびSub_nの詳細な内容は，計測対象になっている個々のクラスや，メッセージなどの進化を定量的に調査することで明らかにできる．

1) システムの進化メトリクス

システム全体の進化を観測対象にしたメトリクスである．次のメトリクスが候補として考えられる．

・全クラス数，全行数

意味）クラス数の変化によって，システムの規模の変化を得る．また，同様にメソッドの行数の総和である全行数もシステムの全体の規模を計測するために使うことができる．

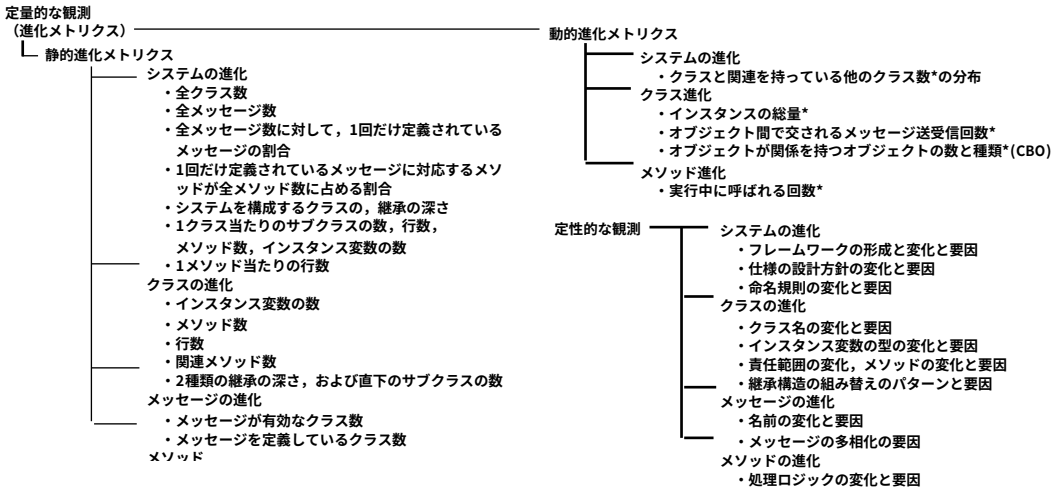

図 1　進化を観測する手段の分類

議論）われわれの計測実験から，ほとんどのクラスについて，1クラスあたりの行数の値がある値に維持される傾向を確認することができた．また，特異な大きな規模を持つ少数のクラスも存在し，これらのクラスの規模の変化がシステム全体の規模に大きく影響を与えていることもわかった．したがって，システムの全体の行数が増加しているからといっても，必ずしも1クラス当たりの行数が増加しているということにはならず，また，クラス数が増加しているということにもならない．全行数の変化は，単にプログラマがコーディングした量の変化を示しているにすぎない．

・全メッセージ数，全メソッド数

意味）全メッセージ数は，ユースケース[9] を実現するためにシステムに定義された機能の種類の数を表し，次の式で表せる．

$$G = \bigcup_{c_i=1}^{n_c} G_{c_i}$$

$$n_g = |G|$$

ここで，G：全メッセージの集合，Gc_i：クラスc_iに定義されているメッセージの集合，n_c：クラス数，n_g：全メッセージ数，である．

メソッド数は，ユースケースを実現するためにシステムに定義された機能の総数を表すメトリクスであり，実際に各クラスに定義されているメソッド数の総和である．したがって，全メソッド数は，次の式で表すことができるメトリクスである．

$$D = \bigcup_{c_i=1}^{n_c} D_{c_i}$$

$$n_d = |D|$$

ここで，D：全メソッドの集合，Dc_i：クラスc_iに定義されているメソッドの集合，n_d：全メソッド数，である．

議論）われわれがここで定義した全メソッド数は，メッセージが定義される回数に依存して変化するメトリクスである．たとえば，すべてのメッセージがシステムの中で1回ずつ定義されている場合，

$$n_g = n_d$$

である．あるメッセージがいくつのクラスで有効か，といった多相性の進化については，メッセージごとの進化メトリクスや，次に挙げるメトリクスを用いて捉える．

・全メッセージ数に対して，1回だけ定義されているメッセージの割合，そのメッセージに対応するメソッドが全メソッド数に占める割合

意味）これらのメトリクスは，メッセージ名の使われ方の変化を観測するためのものである．単に数の増減を議論するのではなく，全体のメッセージ数に対する割合を議論する．

全メッセージ数に対して，1回だけ定義されているメッセージの割合 $p_{g\alpha}$ は，次の式で表せる．

$$p_{g\alpha} = |G_\alpha| / |G|$$

ただし，

$$G = G_\alpha \cup G_\beta$$

ここで，G_α は1回だけ定義されているメッセージの集合，G_β は多重定義されているメッセージの集合を表す．G は G_α と G_β の直和で表わせる．

また，1回だけ定義されているメッセージについて，全メソッド数に占める割合 $p_{d\alpha}$ は，次の式で表せる．

$$p_{d\alpha} = |D_\alpha| / |D|$$

ただし，

$$D = D_\alpha + D_\beta$$
$$|D_\alpha| = |G_\alpha|$$
$$|D_\beta| \geq |G_\beta|$$

ここで，D_α は1回だけ定義されているメッセージに対応するメソッドの集合，D_β は多重定義されているメッセージに対応するメソッドの集合を表す．D は D_α と D_β の直和で表わせる．

議論）同じメッセージのシグネチャを使ったメソッドが定義される回数は，外部変化とシステム内部の事情によって変化する．たとえば，外部変化の可能性としてユースケースの変化がある．同じ仕様が多くの種類のオブジェクトに要求されるような変化が起きた場合は，同じシグネチャを持つメソッドの数が増すだろう．これは，$p_{g\alpha}$ の減少と，それに伴う急激な $p_{d\alpha}$ の減少として観測することができるかもしれない．

また，類似の機能を提供するメソッドが数多く定義されるようになると，個々の違いを明らかにできるようなメッセージの名前が必要になり，外部の仕様変更がなくても，メッセージ名の整理が行われるだろう．その結果，$p_{g\alpha}$ の増加と $p_{d\alpha}$ の維持として観測できる可能性がある．

いずれにしても，$p_{g\alpha}$ や $p_{d\alpha}$ は，詳細な進化のパターンを得るために，定性的にその原因を調査する必要がある．

・システムを構成するクラスの，継承の深さ，サブクラスの数

意味）システムの進化の傾向として，継承の深さとサブクラスの数の変化を捉える．

議論）オブジェクト指向システム開発では，どのように既存のクラスを継承して再利用するかが生産性の鍵を握ると言われている．実際に継承がどの程度用いられているのかを知るためには，ここで提示しているメトリクスを使える．

一般に，継承を定義する場合は，抽象化の作業と具象化の作業が必要である．具象化の作業には，必ずしも抽象化の作業は必要ではないが，抽象化の作業には具象化の作業も必要である．

抽象化と具象化の作業をいつ行うかといった継承の進め方は，定義されるクラスの設計方針によって異なる．再利用クラスライブラリを供給する場合，クラス階層が深くなっても再利用者の要求にできるだけ近いクラスを提供することが重要であるため，具象化と抽象化の作業を並行して実施するだろう．また，一般のアプリケーション開発では，別のプロジェクトでクラスが再利用されることを考慮する必要は少ないため，具象化が主に行われると思われる．その結果，クラスライブラリの継承は深くなる方向に，また，アプリケーション内のクラス階層は浅くとどまり，広くなる方向に進化すると予想できる．

以上のことから，このメトリクスを用いて進化を計測すると，開発されたシステムで用いられた継承の特性を知ることが可能であり，逆に，そのシステムに期待されている特性と，実際にそのシステムに定義されている特性とを比較して，継承を用いた開発の進め方について，その善し悪しを議論することも可能になるだろう．

・システムを構成する1クラス当たりの，行数，メソッド数，インスタンス変数の数，および1メソッド当たりの行数

意味）これらのメトリクスの分布を時系列で観測することによって，システム構成の全体の状態変化を定量化できる．

議論）この分布を時系列で計測することによって，継続的な変化の中から特異な値を抽出し，進化上の課題や注目点を得ることが可能である．そのためには，様々な状況で開発されたシステムの進化を計測する必要がある．

分布は，観測対象のシステムの種類，開発者，開発方法論，開発言語，開発環境などによって影響を受けるであろう．

2) クラスの進化メトリクス

個々のクラスの進化を観測するためのメトリクスである．ここでは継承している親クラスで計測できるメトリクスの値は考慮しないことにした．これを考慮すると，継承されている親クラスの進化が複数の子クラスで観測されることになる．しかし，このような進化は継承構造の影響であるから，継承構造に関係するメトリクスで議論すべき問題である．

ここでは，次のメトリクスが候補として考えられる．

・インスタンス変数の数

意味）クラスの静的な構造の規模の変化を表すメトリクスである．

議論）このメトリクスの値が大きいということは，クラスの状態空間が大きいことを表す．Smalltalk[8] で開発されたシステムでは，インスタンス変数に束縛されるクラスを静的に得ることができない．またC++でも，束縛されるクラスの種類が同じだからと言って，意味的に同じオブジェクトが束縛されるとは限らない．インスタンス変数の数の変化がクラス構造の規模の変化を表すというのは，状態空間の大きさの変化を意味しているのであって，構造そのものの変化をすべて観測できるという意味ではない．

・メソッド数

意味）クラスが担当している役割の数である．

議論）このメトリクスの値が大きいということは，クラスの役割も大きいことを意味する．

・行数

意味）クラスのプログラム上の規模を表すメトリクスで，クラスに実装されているメソッドの総行数である．

議論）行数はプログラミングスタイルによって変化するが，プログラミングスタイルの標準化や，清書化されたプログラムコードを計測対象にすることで，行数を規模の尺度として使用できる．

・継承の深さ，および直下のサブクラスの数

意味）そのクラスの視野を表すメトリクスで，継承の深さは，いくつ位のクラスの変更の影響が自分に及ぶかを，また，直下のサブクラスの数は，自分の変更の影響を直接与える範囲を，それぞれ表している．

議論）アプリケーション開発では，既存のクラスライブラリを再利用して開発する設計と，アプリケーションの機能を拡充していく設計とがある[17]．進化を考えるにあたり，この2つの設計を区別することにした．

ライブラリの再構築までをシステムの進化の範囲に考える場合は前者の設計を考慮して計測を行 い，その進化が設計者が開発したアプリケーションシステムの範囲内に止まっている場合は後者の設計を前提に計測を行う．したがって，継承の深さというメトリクスに基づいて計測を行う場合は，計測対象のアプリケーションの進化の範囲をどこに定めるかによって，深さをどこから計測するかを決める．

CKメトリクスでは，継承の深さを，クラスライブラリまで含めたDIT(Depth of Inheritance Tree)として定義している．このメトリクスは，クラスの理解容易性を計測するために定義されたものである．また，直下のサブクラスの数は，NOC(Number of Children) と呼ばれている．

・関連メソッド数

意味）クラスに定義されているメソッドの総数と各メソッドから呼び出されているメソッドの総数の合計である．

議論）品質の観点からは，このメトリクスはクラスの理解容易性を意味しており，保守性，テスト性，デバッグ性を表すために評価される[4]．進化の観点からは，クラスの複雑度の変化を表す指標として使う．CKメトリクスの一つで，RFC(Reference for Class) と呼ばれている．

3) メッセージの進化メトリクス

システムに求められている機能の多様性と類似性の進化は，メッセージの多相性を計測するメトリクスから求めることができる．これは，要求変更がシステムで，どのように分類され，どのように整理されたのかを定量的に知る手掛かりになる．

各メッセージの進化を観察するためのメトリクスとして，次のメトリクスを候補として考えた．

・メッセージが有効なクラス数

意味）あるメッセージをオブジェクトに送った場合，そのメッセージを処理できるクラスの数である．例えば，あるクラスに定義されているメッセージの集合 G_i があるとすると，そのクラスのすべてのサブクラスがこのメッセージに答えられると考え，次の式によってクラス数を求める．ただし，ここではメッセージの無効化（eg. souldNotImplement など）は考慮しないものとする．

$$n_{cm_i} = \left| \bigcup_{t_i=}^{n_{top}} Ctre_{t_i} \right|$$

$$Ctre_{t_i} = C_{top_{t_i}} + Csub_{t_i}$$

ここで，n_{cmi} ： あるメッセージ i が有効なクラス数，$Ctree_{ti}$ ： メッセージ i を定義しているクラス階層ti上の最上位のクラス$Ctop_{ti}$と，そのクラスのサブクラスの集合$Csub_{ti}$の直和，n_{top} ： メッセージ i を定義しているクラス階層t_i の数，である．

議論）そのメッセージを定義するクラスが増加したからといって，それが継承したメッセージの再定義であれば，そのメッセージ名が有効なクラスの範囲は変化しない．

・メッセージを定義しているクラス数

意味）あるメッセージを定義しているクラス数が増加するということは，語彙に与えられる多様性が増すということである．これらのメッセージの進化を計測するためのメトリクスは，語彙の進化を捉えることが目的である．

議論）メッセージ名は，多相性を考えて定義されるが，多重定義や再定義を繰り返すうちに，一つの語彙が多くの意味を持ち始め，多相性を持つ語彙として開発者が整理し理解できる限界を超えるときがあるのではないかと想像する．

あるメッセージが有効なクラス数の変化とメッセージの多相化を時系列に沿って調査することで，語彙が分化して増加していく過程や，整理されて減少していく過程を捉えることができるだろう．

4) メソッドの進化メトリクス

個々のメソッドの進化を観測するメトリクスで，次の候補が考えられる．

・行数

意味）メソッドの規模を表すメトリクスである．

議論）メソッドの処理手順が複雑化すると，規模も大きくなる．また，クラス階層が変化するに従って，メソッドの再設計が起こることも考えられる．このような進化を，行数の変化から定量的に観測し，ユースケースの変化やクラス階層の変化のメトリクスを用いた計測値と対応づけて捉える．

いつ，どのような条件のときにメソッドの再設計が行われるのかを調査することによって，進化に基づいた設計指針を提示することが可能になるだろう．

3. 進化メトリクスの計測実験

2節で提案した進化メトリクスを用いて計測実験を行った．計測実験の目的は，進化メトリクスによってシステムの進化過程を視覚的に把握できることを検証することにある．したがって，進化メトリクスを用いた計測結果から，どのような傾向が一般に言えるのか，あるいはそれが進化の過程として好ましいか，好ましくないかの議論は，今後の研究になる．

計測実験を行うにあたり，次の条件に合致するアプリケーションシステムを選択した．

1) 開発者が一人，あるいは設計方針が同じチームメンバーで構成されるグループによって開発されていること．今回は進化メトリクスの有効性の調査であるため，版ごとのプログラム構造の変化の原因から，開発者の違いを排除したかった．

2) 少なくとも3版以上の開発記録が入手可能であること．三つの版は，進化の傾向を捉えるために必要である．

3.1. 調査対象の概要

調査対象になったシステムは，Visual Smalltalk上で開発された温度調節シミュレーションシステムである．このシステムは，分析，実装から保守までの8ヵ月を一人の開発者が担当し，その間，顧客に提示したプロトタイプとして６つの版があった．これらのプロトタイプは開発の途中経過として提示されたものであったが，その都度，提示されたプロトタイプを基に，要求の変更および拡張が顧客から提案されていた．したがって，各版を進化の過程と見なしてよいと思われる．

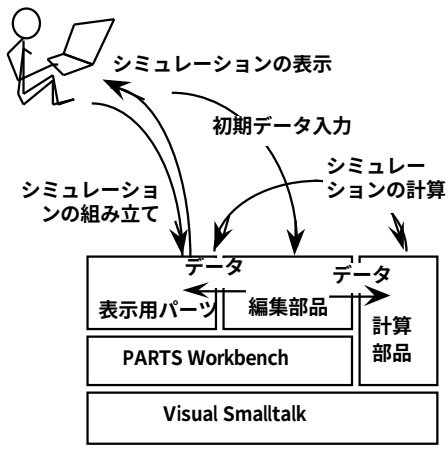

図 2 システム概要図

調査を行ったシステムは，Visual Smalltalkの一部を構成しているPARTS Workbench上に構築された．システムの最終的な基本構成は，シミュレーション対象の設備を視覚的に組み立てる部分である表示部品，シミュレーション用のデータを入力する編集部品，およびシミュレーションを実際に計算する計算部品の三つから成り立っている(図 2)．はじめの二つが，PARTS Workbench上に構築された部分である．システムの規模はver.0でクラス数10，最終のver.5で52になっていた．ver.0からver.1までは2ヵ月，それ以降の版はそれぞれ1ヵ月ごとにリリースされている．各版のリリース内容を次に示す．

1) ver.0

PARTS Workbenchのシミュレーションシステムへの適用可能性を検査するためのプロトタイプ．定義したPARTS Workbench上のシミュレーション用部品を動的に接続し，各部品の特性に従ってシミュレーションを実行できることが確認された．

2) ver.1

シミュレーションを行う最小基本単位として，表示部品と編集部品から構成されるシミュレーションシステムが開発された．

3) ver.2

シミュレーション対象部品が多様化し，さまざまな温度調節のための計算方法が導入された．そのため，シミュレーションの計算を専門に行うオブジェクトが必要になり，この機能を表示部品から独立させた計算部品を定義し，図 2 に示すシステムの基本構造を作った．

4) ver.3

さらに取り扱う部品の多様化，シミュレーションの複雑化が要求されると同時に，複雑化したシミュレーションを簡単なGUIで操作できるようにGUIも変更された．

5) ver.4

シミュレーションのための部品の整理が行われた．

6) ver.5

利用上の問題が解消された．

進化の検討では，ver.1からver.4までを対象とし，ver.0とver.5は除いた．これは，ver.0が使い捨て型のプロトタイプであり，それ以降の進化型プロトタイプとは性質が違うためである．また，ver.5は，計測結果からver.4とほとんど変化が見られなかったため省略した．

3.2. 計測結果

収集したメトリクスを用いた計測結果を示しながら，進化メトリクスの有効性について議論しよう．

1) システムの進化

・全クラス数，全行数，全メッセージ数，全メソッド数の変化

まず最初に，システムの進化の概要を捉える進化メトリクスについて，ver.1からver.4へ至るまでの変化の様子を調査し，図 3 を得た．横軸は1ヵ月単

位の時系列で収集できた版を表し，縦軸は，ver.1における値をそれぞれ1としたときの相対度数を表している．

図 3 から，各期間における開発量の変化を知ることができる．また，メソッド数の変化の様子とメッセージ数の変化の様子に差が観察されるのは，多相性を持つメッセージが定義される量に差があることを意味している．

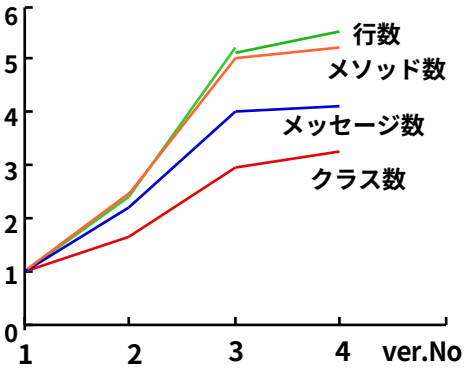

図 3 システムの進化メトリクスを用いた計測結果

ここに揚げた4つのメトリクスを用いた計測結果だけから，必ずしも「システムが進化するにつれて，1クラス当たりのメッセージ数は増加する」とは言い切れない．なぜならば，このシステムでは，後で図 10 に示すように，各クラスに定義されるメッセージ数が増えて行ったのではなく，少数のクラスだけにメッセージの顕著な増加傾向が観測された．すなわち，このメトリクスでは，システムの規模の変化を知ることができるだけで，変化の詳細な内容を捉えるためには，各クラスごとの計測が必要である．

表 1 1回定義されているメッセージの割合の推移

(%)

ver.	1	2	3	4
:メッセージ数	68.6	58.3	59.5	63.3
:メソッド数	43.4	32.9	30.0	31.4

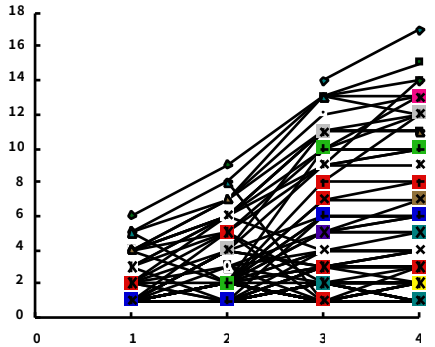

図 4 メッセージを定義しているクラス数の変化の軌跡

・全メッセージ数に対して，1回だけ定義されているメッセージの割合，全メソッド数に対して，そのメッセージに対応するメソッドが占める割合

この進化メトリクスに基づいた計算結果の変化の様子を表 1 に示した．

各版で追加されたユースケースの内容が不明なため表 1 の結果から進化の傾向を議論することはできないが，少なくとも，全メッセージの6割を占めるメッセージが全メソッドの3割程度しか占めていないことは言える．逆に言うと，全メッセージの4割のメッセージは多重定義され全メソッドの7割弱を占めていることになる．

さらに，1回だけ定義されているメッセージは．この変化の内容を詳細に見るために，各メッセージごとの進化メトリクスを用い，メッセージを定義しているクラス数の版ごとの変化を図 4 に示す．

計測実験を行ったシステムの表 1 および図 4 の結果から，メッセージが多重定義される回数が，システムの進化に伴って増加する傾向を観察できる．システムの進化によって定義回数が減少しているメッセージは別のメッセージ名に置き換わったり，一つのメッセージ名がいくつかのメッセージ名に分化する現象が起きたものである．

メッセージ名の進化については，これらの進化メトリクスを用いた計測で視覚化できるが，進化の原因として，定性的にユースケースとの関係を議論しなければ，その詳細な内容を明らかにすることはできない．

・システムを構成するクラスの，継承の深さ，サブクラスの数の変化

図 5 に 継承が定義されていった経過を構造の模式図として示した．また，これを定量的に計測した結果を図 6 に示した．構造的に観察を行うと，どのクラスがいつ定義され，いつ具象クラスが定義されていったか，という情報を得ることができる．ver.1からver.4までの間で，継承されるクラスは色を分けて四角形で示した．このシステムでは，進化の過程で明確な抽象化の作業は1回しか行われていなかった．われわれにとっては，進化メトリクスの評価が目的なので，これ以上の構造的な進化の調査に関する議論は，別の機会に譲ることにする．

さて，図 6 では，横軸に版番号を示し，縦軸には，該当する継承の深さを持っているクラス数を示した．表 2 には，各版における最大の深さと，直下のサブクラスの数の最大値および平均の深さを示した．また，表 4 には，継承の深さの分布を示してある．これらのデータから，今回計測を行ったシミュレーションシステムでは，クラス階層は深くなる方向よりも広くなる方向へ定義される傾向を持っていたことがわかる．これは図 5 の絵の変化からも，直観的に知ることができる．平均の深さも進化によって，あまり変化していない．

・システムを構成する1クラス当たりの，行数，メソッド数，インスタンス変数の数，および1メソッド当たりの行数に注目した分布の変化

典型的な例として，1メソッド当たりの行数と累積度数分布の変化を図 7 に示した．分布は，システムの進化に依存せず，維持されていることがわかる．

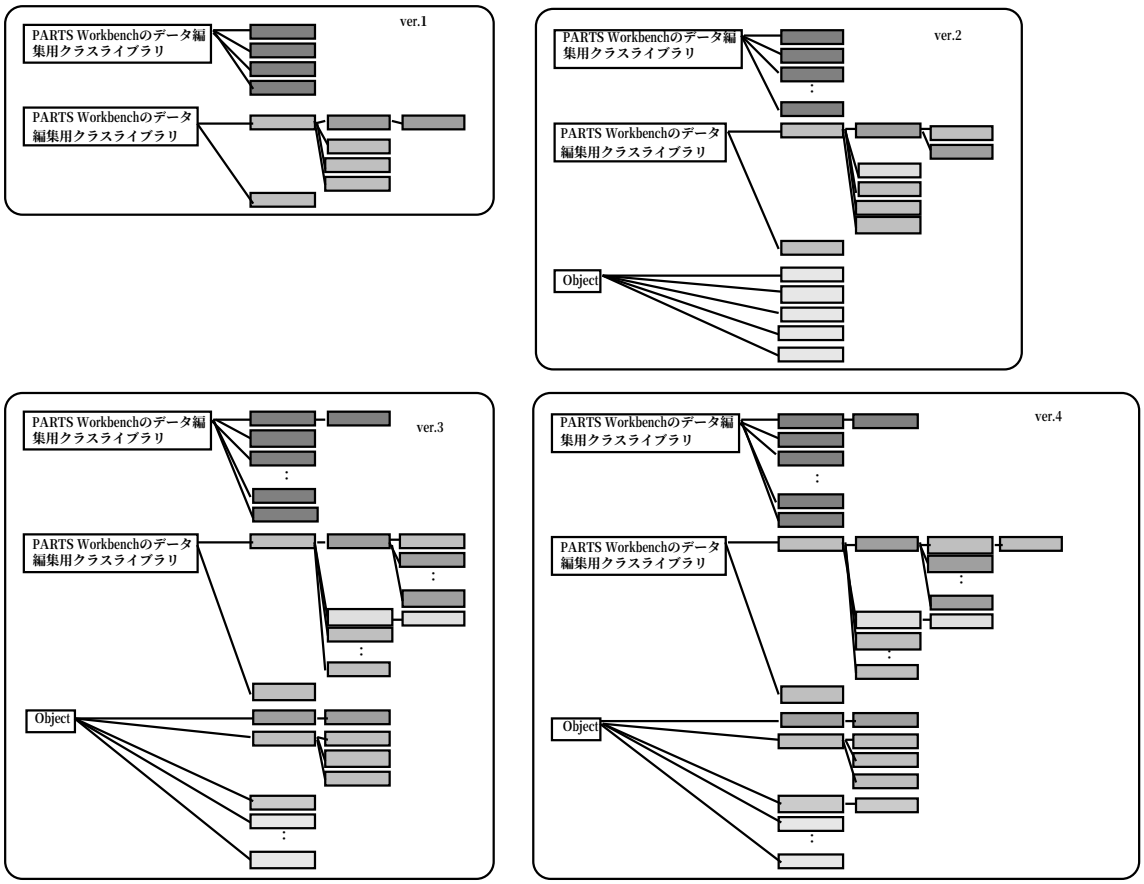

図 5 定性的な継承の進化の調査

□ は，各クラスを表わし，線は継承関係があることを表わす．

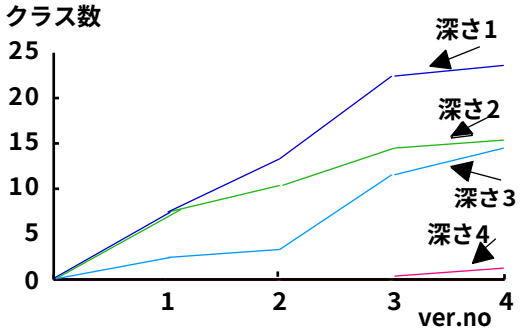

図 6 ライブラリから数えた深さの推移

表 2 継承の深さと幅の推移

ver.	1	2	3	4
最大深さ	3	3	3	4
サブクラスの最大数	4	5	9	10
平均の深さ	1.7	1.7	1.9	2.1

これらのメトリクスには，Kolmogorov-Smirnov 検定[15] などを用いた統計処理を行った結果，次に示す共通の特徴を観測することができた．

(1) システムが進化しても，分布の形状が維持される．

(2) 維持される分布の中の飛び外れ値は，版を重ねるに従って，さらに中央値からの距離を拡大する方向に変化する．ここで外れ値と飛び外れ値は，次のように定義されている．まず，基本統計量から4分位差Dを求め，4分位値から4分位差の1.5倍を刻みとして，1.5D離れた位置を内堀，さらに1.5D離れた位置を外堀と考える．このとき，この内堀と外堀の間にあるデータを「外れ値」，外堀より大きい，あるいは小さいデータを「飛び外れ値」と言う[24] ．

(3) 飛び外れ値を持っている標本やその標本を包含している標本は，他のメトリクスに基づく計測値でも飛び外れ値を表わし，いずれの場合でも特異点として検出される．

開発者へのインタビューを行い，これらのメトリクスに基づいた計測の結果を見せたところ，観測された飛び外れ値を持っていたクラスやメソッドは，システム開発の過程で，その設計が再検討の候補になっていたクラスやメソッドと一致していた．

したがって，進化メトリクスを用いたシステムの進化を観測することによって，設計上の課題を抽出できる可能性がある．設計者の考えだけに留められていた設計上の課題を，進化メトリクスから明らかにできるということは，定量的な設計支援の可能性

があることを意味している．

・進化メトリクスの加工：メトリクスの相関

ここまでに述べた，進化メトリクスを用いた計測結果の分布傾向に類似点が観察されたので，これらの進化メトリクスの間の相関分析を行ったところ，クラスの行数，インスタンス変数の数，メソッド数のそれぞれの間に強い正の相関関係が認められた．

クラスのインスタンス変数の数はクラス構造の規模を表しており，メソッド数はクラスの役割の数を表している．また，クラスの役割はクラスの構造によって実現される．したがって，クラスの役割がその構造によって制約されるのは自然である．また，行数は，役割を物理的に実現した規模であるから，役割の数と行数に相関関係があることにも納得がいく．そこで，行数とメソッド数に関する各版の進化について，散布図を描いてみた．図 8 にその結果を示す．各版を見比べると，初期の版で規模の小さいクラスが定義され，版が重ねられるに従って規模の大きいクラスが定義されていった開発プロセスを知

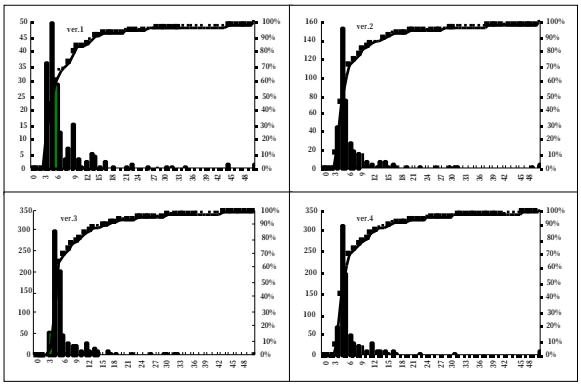

図 7　1メソッド当たりの行数の
ヒストグラムとその累積度数分布のの変化

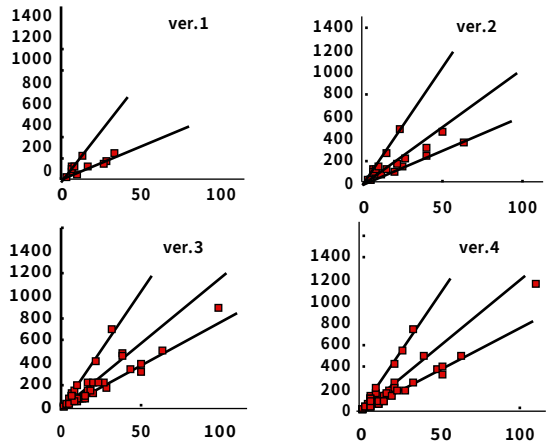

図 8　行数とメソッド数の相関関係の進化

表 3 クラスのメソッド数と行数の回帰係数

ver.	1	2	3	4
表示部品	6.80	7.93	8.69	9.31
編集部品		8.39	19.88	20.07
計算部品		5.67	6.23	6.20

ることができる．これは，インクリメンタル開発の軌跡であると考えることができる．

さらに，散布図には，3方向に延びる線を観察することができる．この各線には，図 2 の基本構造を構成する三つのクラス階層が対応している．この直線の傾きを求めるために回帰分析を行った．その結果を表 3 に示す．各クラス階層ごとに固有の値があるように見える．

図 9 には，ひとつのクラス階層に属するクラスだけを取り出して，各版の散布図を重ね，個々のクラスの進化を線で結んだ．図 9 で観察できる直線上に乗っていない1点は，ver.3だけでここにプロットされた．このクラスはver.4で再設計が行われ，直線上に乗る値を示すように変化していた．開発者は，進化メトリクスを用いて進化の計測を行っていたわけではなく，開発者が直観的に捉えていた設計上の問題点を，このようなメトリクスを用いて求めることができたことになる．したがって，ここでも，定量的な設計支援の可能性があることがわかる．

すべてのシステムでこのような直線関係が成立するかどうか，あるいは複数の開発者が関わった場合はどのような傾向を示すのかを明らかにするためには，さらに多くのデータを収集する必要がある．

2) クラス，メッセージ，メソッドの進化メトリクス

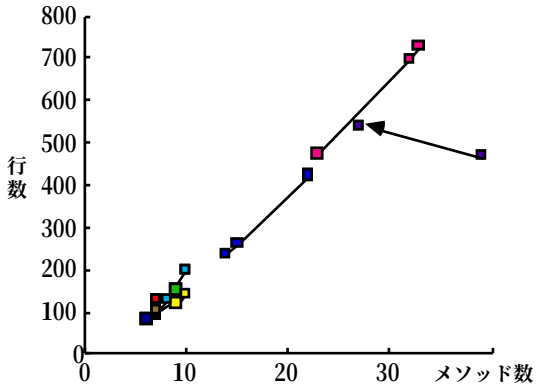

図 9　クラスのメソッド数と行数の相関関係
（クラス階層Aの各版を色別して表示してある．ほとんどのクラスが直線上に乗っている．同時に異常値を発見することもできる）

表 4 継承の深さの推移

ver.	1	2	3	4
平均	1.7	1.7	1.9	2.1
中央値	2	2	2	2
最大	3	3	3	4
最小	1	1	1	1
データ数	16	26	47	52

これらのメトリクスによって，個々のクラスやメッセージ，メソッドはどのような進化を経るものかを知る手がかりになっただけでなく，システムの進化メトリクスを用いて計測した結果から，たとえば飛び外れ値を持っていたクラスやメソッドが同じものなのか否かを追跡できた．

図 10 に，システムの進化によって変化するシステム構成要素の変化の様子を示した．この図は1クラス当たりのメソッド数が，システムが版を重ねるごとにどのように変化していったかを示したものである．個々のクラスがそれぞれ1本の軌跡を描いている．横軸は版の番号を示し，縦軸は1クラス当たりのメソッド数を表している．

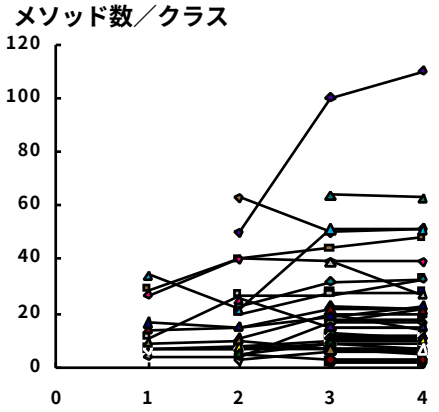

図 10 1クラス当たりのメソッド数の変化の軌跡

この図から，観測を行ったシステムでは，進化メトリクスを用いて観測されたメソッド数の増加が，特定のクラスで定義されるメソッド数の増加の影響を受けていることがわかった．また，飛び外れた値を持っていしまうと，進化によって，さらにその傾向を大きくしていく傾向があることも観察できる．

4. 進化メトリクスの評価

本研究では，進化メトリクスによって進化を捉える可能性と有効性を調査するために，1種類のアプリケーションについて計測実験を行った．計測実験から進化メトリクスの特徴がいくつか明らかになったが，実験対象が1つだけであるため，オブジェクトの進化の特徴を議論することは避けたい．

ここでは，オブジェクトの進化を計測する尺度の有効性について議論することにする．進化メトリクスを用いて，オブジェクトの進化を計測する意義を次にまとめた．

1) システムの進化メトリクスに基づいた計測では，計測値の分布形が時系列上で維持される傾向を確認できた．また，時系列で収集した観測標本の中には，その分布が一定の範囲内に維持されているものと，維持されず中央値から外れていく傾向を示すものを観測できた．さらに，このような標本は，開発者が「設計上，問題のある部分」として抽出していたものと一致していた．

以上のことから，進化メトリクスを用いることは，設計上の課題を定量的に抽出するために有効である．

2) 時系列で計測した値の変化から，開発のライフサイクルモデルを捉えることができた．

計測実験の結果では，一様な規模のクラスが一定の割合で開発されて行ったのではなく，規模の小さなクラスから開発が始まり，次第に規模の大きなクラスが定義され，その後，規模の大きいクラスが成長していくという軌跡を捉えることができた．われわれは，システムの進化のいくつかのパターンを得ることができれば，システムの寿命や拡張計画を定義できるのではないかと考えている．

したがって，現状では，まず様々な開発の軌跡を捉え，進化のパターンを得る必要がある．われわれが提案した進化メトリクスは，開発の軌跡を捉えるために有効である．

3) クラス階層による層別にデータを整理することで，クラス階層ごとの特徴を捉えることができた．そして，クラス階層ごとに明らかな有意差が存在することが明らかになった．この結果から，オブジェクトの品質や適／不適を議論する場合は，一般的なオブジェクト指向システムの計測値と比べるよりも，クラス階層が持っている固有の値と比べて議論した方が適切であることがわかった．

クラス階層を統計処理の層として取り扱うことによって，クラス階層に固有の値を得ることができれば，保守や最設計の指針として取り扱うことが可能になる．

5. 関連研究と今後の展開

オブジェクト指向メトリクスの分野では，複雑度，凝集度，理解容易性，および保守性を定量化するためのChidamberらのCKメトリクス[4] や，プロジェクト管理に利用する生産性などを計測するLorenzらのメトリクス[13] などが提案されている．また，再利用の効果を計測するためのメトリクスの検討も中西らによって行われている[16] ．このようなオブジェクト指向メトリクスは1990年代に入ってから，多くの研究が進められてきた[23] ．われわれは，システムに共通な特徴を捉えることよりも，開発対象の特性を考慮した上で進化の適切さを議論することに重点をおいた進化メトリクスの提案を行った．

開発対象の特性を考慮するためには，そのシステム

の進化に従って，メトリクスに基づいた値を計測し，進化による維持と変化を捉えなければならない．今後は，他のアプリケーションに対して進化メトリクスを適用し，システムの特徴ごとの進化パターンの分類を行い，ライブラリの進化との差についても議論しなければならないと考えている．

また，今回の研究では，定性的な進化についてはほとんど触れていない．クラス構造の組み替えのパターンは，文献[1] [2] [11] など，オブジェクト指向データベースのスキーマ進化の研究で提示されている．また，とくに継承構造の構築パターンについては文献[19] で議論されている．今後は，システムの進化とこれらのスキーマ進化のパターンとを，システムに要求されるユースケースの変化と関連づけて議論していく必要もあるだろう．

システムの拡張によるクラス構造の変化に対して，プログラミング言語では，送られたメッセージとコンテキストを提示して随時適切なメソッドを起動させるような言語仕様も研究されている．システムが拡張されると，それまで定めていた視点が無効になり，クラス構造の再設計を行わなければならないことがあるが，文献[6] の研究は複数のアプリケーションに再利用されるクラスに，各アプリケーション毎の仕様を提供する仕組みを検討している．

システムの再設計や拡張を支援する環境も提案されている[18] が，メトリクスを用い，視覚的にシステムの過去と現状と将来の指針を提示する開発支援環境の開発は，今後のわれわれの研究課題である．

謝辞

本研究は，通商産業省ならびに情報処理振興事業協会の推進する独創的情報技術育成事業の一環として行われたものである．研究の機会を与えてくださった同事業関係者の皆様に感謝いたします．

参考文献

[1] Bergstein, P. L. & Lieberherr, K.J. , "Incremental Class Dictionary Learning and Optimization," *Proc. of ECOOP'91*, Springer Verlag, Geneva, Switzerland, 1991, pp.377-396.

[2] Bergstein, P.L, "Object-Preserving Class Trans-formations," *OOPSLA'91*, ACM, 1991, pp.299-313.

[3] Boehm, B.W. *Software Engineering Economics,* Prentice Hall, 1981.

[4] 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.

[5] ダーウィン，C.著，八杉龍一訳，*種の起原*，岩波文庫, 1990 (1859).

[6] Erradi, M., Bochmann, G. v. & Dssouli, R., "A Framework for Dynamic Evolution of Object-Oriented Specifications," *Proc. of the Conference on Software Maintenance*, IEEE, November 1992, pp. 96 - 104.

[7] Gamma, E. et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, MA, 1994.

[8] Goldberg, *Smalltalk-80 : The Langage*, Addison-Wesley, 1984.

[9] Jacobson, I., et al., *Object-Oriented Software Engi-neering*, Addison-Wesley, 1992.

[10] Lehman, M.M. & Belady, L.A. *Program Evolution*, Academic Press, 1985.

[11] Lerner, B. S. & Habermann, A. N., "Beyond Schema Evolution to Database Reorganization," *Proc. of ECOOP/OOPSLA '90*, 1990, pp.67-76.

[12] Lientz, B. P. & Swanson, E. B., *Software Mainte-nance Management*, Addison-Wesley, 1980.

[13] Lorenz, M. & Kidd, J. *Object-Oriented Software Metrics*, Prentice Hall, 1994.

[14] Mayr, E., *One Long Argument, Charles Darwin and the Genesis of Modern Evolutionary Thought*, Harvard University Press, 1991. (養老孟司訳, ダーウィン進化論の現在, 岩波書店, 1994).

[15] 森口繁一, "確率表現関数の検定について", *日本統計学会誌*, 第25巻, 第3号 (1995), pp.233-244.

[16] 中西弘毅 & 荒野高志, オブジェクト指向プログラミングにおけるクラスインタフェースの抽象化と利用効果の評価メトリクス, *サマーワークショップ・イン・立山予稿集*, 情報処理学会, pp.145-151.

[17] 中谷多哉子 & 春木良且, "オブジェクト指向における効果的な部品の再利用," *ソフトウェアシンポジウム'93予稿集*, 1993, pp.6-14.

[18] Ossher, H. & Harrison, W., "Support for Change in RPDE3," *SIGSOFT*, ACM, 1990, pp. 218-228.

[19] Ossher, H. & Harrison, W., "Combination of Inheritance Hierarchies," *Proc. of OOPSLA'92*, ACM, 1992, pp. 25-40.

[20] Rumbaugh,J., et al. *Object-Oriented Modeling & Design*, Prentice Hall, 1991.

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

[22] 鳥光陽介，玉井哲雄, "ソフトウェア・システムの寿命とその要因についての考察," *ソフトウェアシンポジウム'92予稿集*, 1992, pp.E-2-10.

[23] Whitty, R. ''Object-Oriented Metrics: A Status Report'', *Object Expert*, Vol.1(2) (1996), pp.35-40.
[24] 吉澤正, *統計処理*, 岩波書店, 1992.