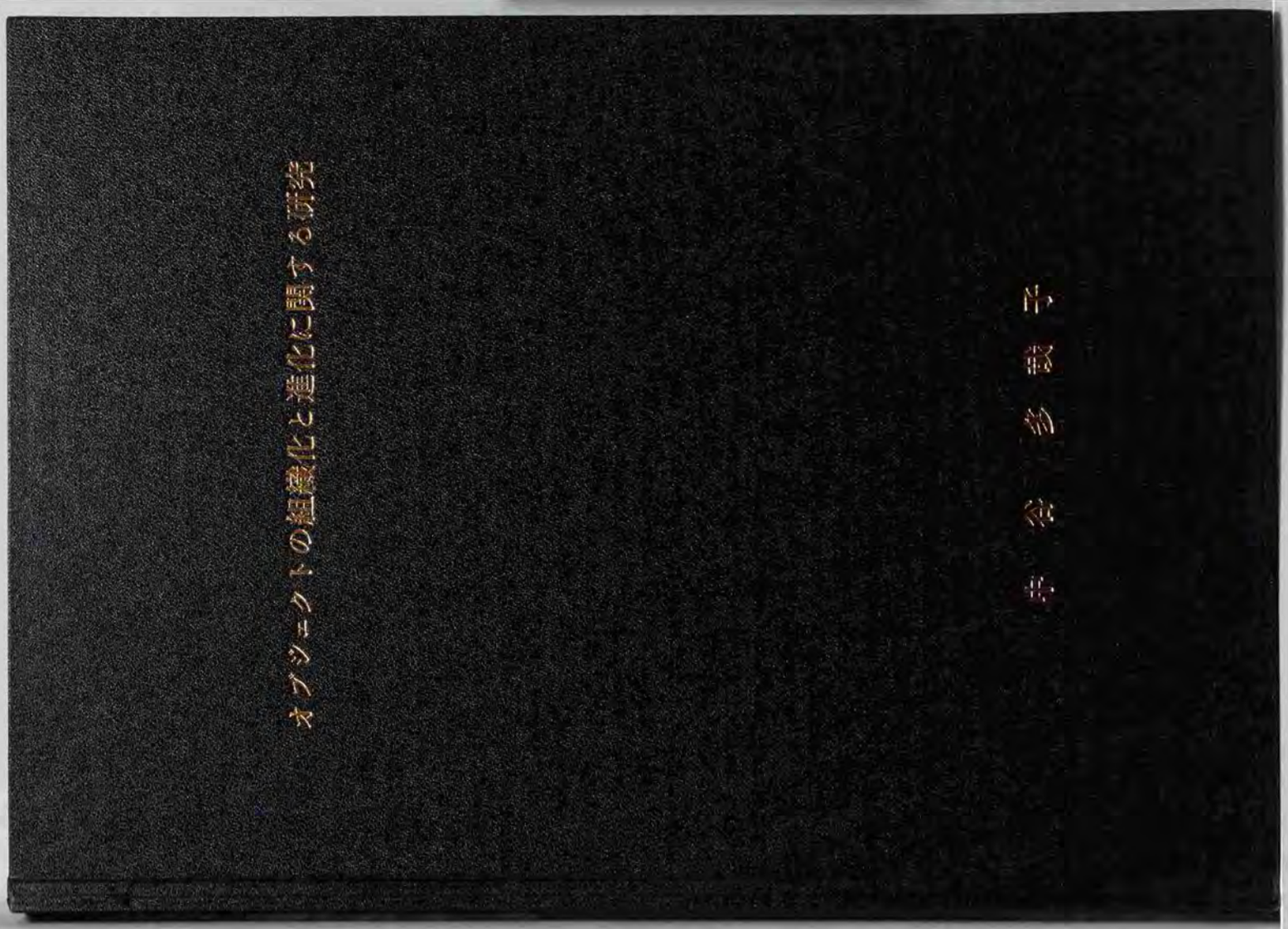
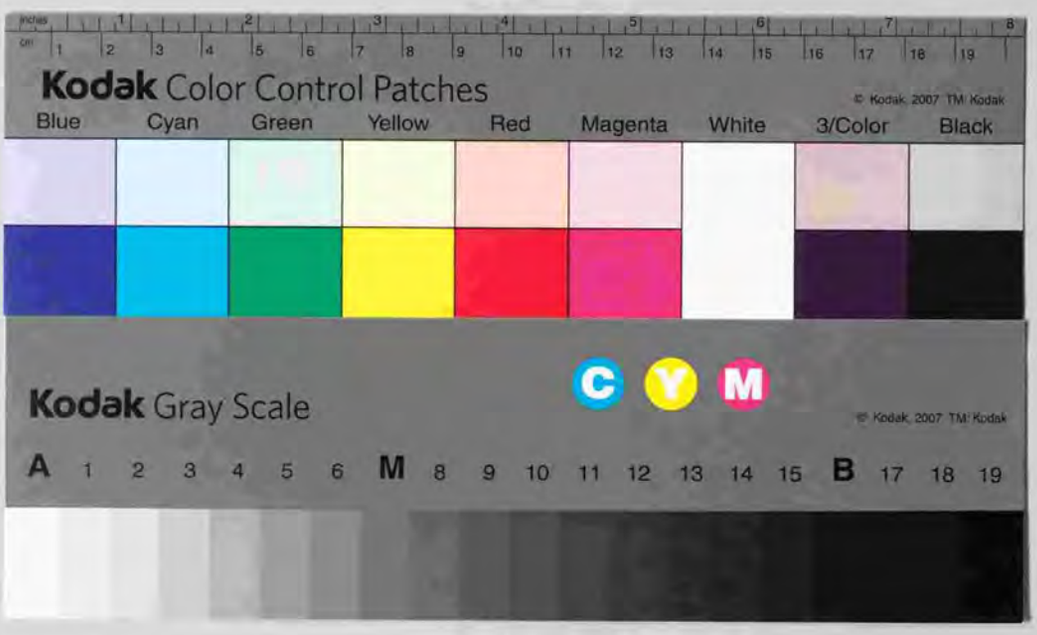

オブジェクトの組織化と進化に関する研究
牛越知哉


Kodak Color Control Patches
© Kodak, 2007 TM Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
© Kodak, 2007 TM Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19

オブジェクトの組織化と進化に関する研究

Modeling Object Organization and Evolution Processes

1998年2月

中谷 多哉子

もくじ

図一覧

表一覧

序論

本論文では，オブジェクトが定義され，変更される過程に生物学のアナロジを適用し，オブジェクトの組織化過程と進化過程について議論する．議論を始める前に，本研究の背景と目的について説明する．

オブジェクト指向技術は1990年中期以降，急速に企業での導入が進められた[24]．企業への導入が促進された要因として，オブジェクト指向自身が提供する新しいシステム開発の枠組み，分析／設計方法論，そして新しい再利用形態の提案を挙げることができる．もともとオブジェクト指向が提供する継承や集約といった機構は，生産性やシステムの信頼性を向上させるための技術として注目されていたが，1980年代後半から紹介され始めたオブジェクト指向による分析および設計方法論[7, 9, 29, 50, 51, 55, 62]も第二世代に入り，CASE (Computer Aided Software Engineering) が実用レベルに達しつつあるといった背景が，企業のオブジェクト指向導入に拍車をかけたようである．また，1990年代初めまで，われわれが再利用できる単位として取り扱ったオブジェクトはクラスだけであったが，1990年代中期に，より大きな単位でクラスを再利用する方法や，抽象度の高いレベルで再利用される単位が提案され始め，再利用が現実味を帯びてきた．とくにフレームワークは，協調動作するクラスをメッセージのプロトコルとともに定義し，複合化されたクラス群をひとつの単位として再利用するためのオブジェクトであり，抽象度の高い再利用を可能にした[34]．また，Gammaらが提案した設計パターンやCoadらがまとめたオブジェクトモデルのパターンは，開発者が蓄積してきた設計や分析のノウハウを再利用するための単位として注目され，1990年代後半のオブジェクト指向開発に大きな影響を与えた[16, 38, 10, 11, 15]．さらに，実装レベルでも新しい形式の再利用が実用化されている．コンポーネントと呼ばれる再利用部品は，稼働環境上で再利用するインスタンスである．プログラマはコンポー

ネントのインタフェースを参照しながら視覚的なプログラミングを行う．そのためコンポーネントウェアと呼ばれる製品群には，視覚的プログラミングを支援する再利用環境も提供されている．これらの製品は，すでに実用レベルの実績もあげられ，ビジネスアプリケーションへの適用も進められている [1]．

このように企業へのオブジェクト指向の導入が促進されると，今後，少なからぬオブジェクト指向ソフトウェアがエンドユーザへ提供されるようになるであろう．ある種のソフトウェアシステムは製品提供後も利用者の要求変更に対応するための継続的な開発を避けることができない．Tate は論文の中で "....evolutionary Prototyping, in the sense that the prototype evolves into the production system should, and perhaps will in future, simply be called evolutionary development" と言っている [58]．また，ソフトウェアシステムが利用者要求変更に適合し続けるためには，漸進型開発形態が適しているとも言われている [5, 12]．漸進型開発形態は，オブジェクト指向でも自然な開発形態として導入されているが [20]，分析／設計方法論や再利用に関する研究は，いずれも新規システム開発のための提案であり，漸進的な開発では，その第一段階だけが支援されているにすぎない．これからのオブジェクト指向ソフトウェア開発では，初期開発後のシステム開発過程に着目した研究がもっと重視されるべきであろう．

システムが漸進的に開発されるとき，オブジェクト指向システム開発者は新しい利用者要求に対応するためにクラスを変更し，同時に次の開発へ向けて，クラスを拡張容易な構造にするといった設計変更も行う [6]．このようなクラスの設計変更をクラスの側から観察すると，開発状況や利用者の要求変更といったクラスを取り巻く状況の変化に対応して，クラスがその構造や仕様を変更させているように見える．ここで生物学のアナロジを適用すると，クラスは，それを取り巻く環境変化に適応するために進化すると言うことができる．

本研究では，漸進型開発に焦点を当て，クラスの仕様やクラス継承木，システムの進化を定量的に観測し，それらの

1. 進化過程と設計意思決定の関連づけ
2. 進化パターン
3. ソフトウェアアーキテクチャの評価手段としての進化モデルの有効性

を示すことを試みた．

本研究では，オブジェクトの進化過程を定量的に表現する手段として進化メトリクスを定義し，実システム開発事例に適用することで，システムの開発に沿った時系列の計測データを収集してオブジェクトの進化過程を分析した．定量的な分析では，あわせて定性的な分析や開発者へのインタビューも行った．

本研究のもうひとつの課題は，オブジェクトが組織を形成する過程に関する研究である．オブジェクトの進化を考慮した組織化は，どのように進めるのが好ましいのであろうか．ここで再び生物学のアナロジを用いて，進化に適したオブジェクトの組織化の過程を考える．生物は，過去の原始的な生物から現在に至るまでに経た進化の過程を，組織を形成する際に再現すると言われている．オブジェクトがシステムの１要素として機能する組織を形成する際にも，生物のような組織化の過程を適用することができる．原始的なオブジェクトを，利用者やシステムに必要な機能や構造を保持していないオブジェクトと仮に定めよう．オブジェクトが進化する主な原動力は利用者要求であるから，オブジェクトを組織化する際にも，利用者要求をひとつづつオブジェクトの組織に取り込ませることが可能である．本研究では，利用者要求を個々に独立した視点で定め，それぞれの要求を満足するオブジェクトの組織を定義し，最終的に個々の組織を統合してシステム全体で協調するオブジェクトの組織を完成させる手法を提案する．本論文では，多視点を用いたオブジェクトの組織が形成される過程を組織化過程と呼び，従来の分析方法論の拡張を試みる．この手法では，システムに将来要求される利用者の要求を完全に予知することは不可能であるとの認識から，多面的に検討した構造をオブジェクトに定義することによって，単一の視点からモデル化するオブジェクトよりも新しい要求への適用範囲を広げることを目指した．新しい要求への適用範囲が広い組織は，オブジェクトの進化にも適した組織となると期待される．

本論文の第1章では，進化の分析結果を説明するための用語であるオブジェクトの進化，そして進化を定量化するための手段として進化メトリクスを定義する．また，調査対象としたシステムの概要もこの章で紹介する．第2章では，3つの実システムに対して進化メトリクスを適用した結果から，オブジェクトの進化の特徴を捉え，定量的な進化に関するデータに統計モデルを当てはめることを試みる．第3章では，クラスの継承木に着目し，オブジェクトの進化に

よって変化しない性質について仮説を提起し，検証する．第4章では，クラスを種に分類し，システムのソフトウェアアーキテクチャに基づいて想定される種の進化パターンと観測された個々のクラスの進化パターンとを比較し，オブジェクトの進化の要因とソフトウェアアーキテクチャの妥当性について議論する．第5章では，オブジェクトの組織化過程について議論し，事例に適用することで，その有効性を検証する．最後の第6章では，関連研究を示すとともに，本論文で議論したオブジェクトの進化過程やオブジェクトの組織化過程の解決すべき今後の課題について議論する．

謝辞

本論文の執筆にあたり，筑波大学大学院経営システム科学専攻在学中より懇切なるご指導をいただきました東京大学大学院総合文化研究科玉井哲雄教授に深く感謝いたします．また，同研究科の川合慧教授，山口和紀助教授，山口泰助教授にはKTYYゼミを通して，数多くの助言をいただきました．またKTYYゼミでは，共に研究に励む学生の方々からも，研究の新しい観点を示唆していただきました．皆様には心から感謝いたします．

また，本研究には，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成果が含まれています．本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．

本研究を進めるにあたり，実システム開発における開発データが必須でした．実開発データの収集では，酒匂寛氏（当時，株式会社SRA），友枝敦氏（当時，株式会社SRA）ほか，株式会社SRAの技術者の皆様にご協力をいただきました．とくに，Smalltalkerである西中芳幸氏，近藤博次氏，松田晴美氏には，多忙のなか，時間を割いてインタビューの時間を取っていただきました．皆様のご協力なくしては本研究をこのような形で完成させることはできなかったと思います．深く感謝いたします．

有限会社インアルカディアの藤野晃延氏には，オブジェクトの組織化過程を適用したモデルのレビューで有益な助言を数多く頂いただけでなく，精神面で多くの叱咤激励をいただき，研究とコンサルテーション業務の両立の面でも支えていただきました．また，株式会社情報技術コンソーシアムの長島一夫氏には，モデル化手法を精緻化する機会を与えていただきました．皆様にも，この場を借りて感謝の意を表します．

最後になりましたが，本研究を進める上で，研究姿勢や将来の夢を共に語り合い，多くの励ましと力を与えて下さった湘南短期大学神林靖先生，慶應義塾大学大学院滝本宗宏氏に深く感謝申し上げます．

第 1 章

進化と環境

1.1 概要

システムが漸進的に開発される際，新しい利用者要求へ対応するために実際に変更されるのは，システムの中に定義されているクラスである．このとき開発者は，単に求められた新たな要求を満足させることを考えるのではなく，将来必要となるクラスの再利用性や拡張性を考慮した設計を行うことが多い．ここで，利用者の要求や開発者の設計意図といったオブジェクトの変更要因をオブジェクトの環境と考えると，オブジェクトは，これらの環境に対して適切な構造や仕様を持つように変更されているとみなすことができる．オブジェクトのこのような変化に対して，生物学のアナロジを適用するならば，オブジェクトは環境の変化に適応して進化すると言うことができる．

以下で，オブジェクトの進化のための環境，そして進化を定量化するための進化メトリクス，および進化率メトリクスを定義した後，本研究で用いた 3 つのシステムの概要と，進化メトリクスを適用した結果の一部を紹介する．

1.2 オブジェクトを進化させる環境

本研究では，オブジェクトを進化させる環境として次に示す二種類の環境を考える．

1. 利用者環境

M. Lehman と L. Belady は，利用者と相互作用を持つシステムを E 型システムと呼んだ [32]．このようなシステムでは，システムが利用者に提供されること自体が利用者の作業環境に変化をもたらし，新たな要求を発生させる原動力となる．そして，新たな要求を受け取ることによってシステムは再び進化する．これを利用者とシステムの相互作用と言う．本研究で調査対象としたシステムも E 型システムである．そこで，オブジェクトを進化させる「新しい要求がシステムに到着する」という事象が利用者の要求変化に由来する点に着目し，この事象が発生するオブジェクトの環境を利用者環境と呼ぶことにする．

2. 技術的環境

利用者環境が変化した際，開発者は，新しい要求に合致するようにオブジェクトを変更するだけでなく，将来の漸進的開発の生産性についても配慮した設計変更を行うことがある．たとえば，与えられた利用者環境の変化や将来予想される利用者環境の変化に対して，既存のクラスを拡張する，分割して新たなクラスを定義する，あるいは統合する，継承構造を変更するといった設計方針を決定する開発者の技術的な意図は，いずれもオブジェクトの進化に影響を与える環境となり得る．そこで，開発者の設計方針が変化するという事象が開発者の技術的意思決定に由来する点に着目し，この事象が発生するオブジェクトの環境を技術的環境と呼ぶことにする．

1.3 進化メトリクス

オブジェクト指向メトリクスに関しては 1990 年以降活発に研究が進められてきた [61]．特に Chidamber と Kemerer らが提案したオブジェクト指向メトリクス（以降 CK メトリクス）[8] は，以降のオブジェクト指向におけるメトリクスの研究に大きな影響を与えた．たとえば，Basili らはエラー発生度と CK メトリクスを用いた計測値との相関関係を示し [2]，Sharble らは，方法論の違いによる成果物を複雑度で比較する際に，CK メトリクスを適用した [52]．

また，Lorenz と Kidd らは，プロジェクト管理および設計の視点からメトリクスを定義し [37]，生産性や見積りへの適用，設計妥当性評価への適用を提案

している．Henderson-Sellers らは，従来のサイクロマティックバリューやファンクションポイントといったシステムの複雑度を評価するメトリクスを，オブジェクト指向へ適用する研究を行った [22]．このようなメトリクスに関する基礎的な研究が進むなかで，再利用率を定義してクラスの再利用率に対する費用対効果のシミュレーションを行った結果も報告されている [21]．中西らは，オブジェクトの操作の抽象化に要する労力と，それによって得られた効用を定量化するメトリクスを提案し，ET++，InterViews を用いて，改版による労力と効用の関係が変化する様子を時系列で示した [41]．

プロジェクト管理における見積り，品質管理へのメトリクスの適用は従来から進められてきた [13, 30, 25]．プロジェクト管理では，計測した結果を解釈し，スケジュールの見直しやリソースの再割り当ての意思決定を行うためにも使われている [56, 60]．

Lorenz と Kidd らはメトリクスの取扱いに関して，一般的な計測値と，ある時点のシステムの状態を計測した結果を比較してシステムやプロジェクトを評価する指針を示しているが，このような評価の方法には疑問がある．個々のプロジェクトが置かれた状況や対象問題領域の特徴も異なるため，開発中のシステムの評価を一般的な傾向と比較して評価するのは適切ではないだろう．本研究では，同一のシステム開発に対して，時系列で計測を行うことによって，その変化の傾向をオブジェクトやシステムの進化として捉え [44, 43]，その進化過程から設計上の問題点を発見することを試みた．進化過程を観測する対象には，次に挙げる 3 つのレベルを設定する．

1. システムレベル

システムレベルでは，以下に示すメトリクスを用いた計測値の他に，システム内の全クラスや全メソッドに対して計測した値の分布や分布の変化の傾向を参照することで，システムの進化を捉えることが可能である．システムレベルの計測結果は，オブジェクトの進化過程を議論する際，システムがどのような進化過程にあったかを確認するためにも利用した．

- NCD: クラス数 (the number of classes)
- SLOC: 全クラスの行数をシステム全体で合計した値 (system lines of code)

- SNOM: 全クラスに定義されているメソッド数をシステム全体で総計した値 (the number of methods defined in the system)
- NMN: システムに定義されたメッセージの名前の総数 (the number of message names defined in the system)
- NCT: ライブラリクラスの直下に定義されているクラス継承木の数. (the number of class trees)
- NCBT: クラス継承木に属しているクラスの数 (the number of classes belonging to a class tree)

2. クラスレベル

Chidamber らは，クラスの可読性，変更容易性，結合度，凝集度を計測するためのメトリクスとして WMC (Weighted Methods Per Class)，DIT (Depth of Inheritance Tree)，NOC(Number of Children)，CBO(Coupling between object classes)，RFC(Response For a Class)，そして，LCOM(Lack of Cohesion in Methods) を定義した [8]．本研究では，オブジェクトの進化を定量的に表現できることを評価基準とし，CK メトリクスのうちの 4 つのメトリクスを進化メトリクスに取り入れた．

クラスの役割の変化を定量化するためには，WMC の代わりにクラスのメソッド数:CNOM を用いる．CNOM は，メソッドの重みづけを 1 とした場合の WMC と考えることもできる．

また，DIT については，直近のライブラリクラスから求めた継承の深さと定義する．CK メトリクスでは，DIT を最上位のクラスから数えた継承の深さと定義しているが，クラスの進化を議論する場合には適切ではない．DIT を用いてクラスの進化を議論する際には，技術的環境の変化が及ぶ範囲に着目して，その範囲の継承の深さを計測すればよいはずである．一般に，アプリケーションの開発者がクラスの継承木を変更する際，ライブラリクラスの仕様を変更することが許されていることは少なく，開発者が自ら定義したクラス継承木だけを設計変更の対象とすることが多い．したがって，進化メトリクスで用いる DIT では，深さの起点をアプリケーションのクラスから見た直近のライブラリクラスと定める．

クラスの結合度を定量化するメトリクスである CBO は，プログラムの稼働中に収集したプロファイルデータからオブジェクト間のメッセージ送受信の回数を集計することで，クラスではなく，インスタンス間の CBO を求めることができる．計測対象システムが Smalltalk のように型のないプログラミング言語で開発されている場合，ソースコードから求めることは困難であるため，CBO は，インスタンス間で交されるメッセージ送受信回数は動的メトリクスとして定義する必要がある．

- CLOC: クラスに定義されている全メソッドの行数をクラス内で合計した値 (class lines of code)
- NIV: クラスに定義されているインスタンス変数の数 (the number of instance variables)
- CNOM: クラスに定義されているメソッド数 (the number of method definitions)
- DIT: 直近のライブラリクラスから数えた継承の深さ (depth of the inheritance tree)
- NOC: 直下のサブクラスの数 (the number of children)
- 動的メトリクス：
  - NSM: 送信したメッセージ名と送信先別の送信回数 (the number of sent messages)
  - NRM: 受信したメッセージ名と発信元別の受信回数 (the number of received messages)

3. メソッドレベル

クラスの役割を表すメソッドの規模は，クラスの規模にも大きな影響を与える．クラスの行数は，ここで定義されるメソッドの行数:MLOC をクラス内で合計した値である．行数の数え方については，Lorenz らが設計を評価するためのメトリクスの中で議論しているように [37]，メッセージ送信文の数を行数とするものなどがある．本研究で計測した行数は，計測方法が簡単な改行の数と定義し，次のメトリクスを用いてオブジェクトの進化過程を定量化することにした．

- MLOC: メソッドの行数 (lines of code of a method)

本論文では，オブジェクトの進化とは，主にクラスの進化を指し，メッセージの進化および動的メトリクスを用いたオブジェクトの進化は，本論文の対象外となっている．また，計測対象システムは Smalltalk で開発されていたため，Smalltalk のメタクラス機構を用いて計測を行うことができた [31]．

1.4 進化率メトリクス

クラスの進化を解析する際，クラスの役割に着目したクラスの時系列の進化率を用いる．クラスの進化率を以下に定義する．

$$\rho CNOM_i = \frac{CNOM_i - CNOM_{i-1}}{CNOM_{i-1}} \log_{10} CNOM_{i-1}$$

ここで，$CNOM_i$ は，第 i 版のクラスのメソッド数を表す．役割に着目したクラスの進化率は，第 i 版で計測したメソッド数 $CNOM_i$ からひとつ前の版のメソッド数 $CNOM_{i-1}$ を引き，その値を $CNOM_{i-1}$ で割った後，$CNOM_{i-1}$ の $\log_{10}$ を掛けた値と定義する．

$\log_{10}$ を乗算しているのは，$CNOM_{i-1}$ の値が小さいとき全体の変化率が大きくなるのを防ぐためである [45]．

1.5 調査対象システムの概要

オブジェクトの進化を調査する対象として選択したシステムは，次の 3 システムである．

1. SystemA: 熱交換シミュレーションシステム

   SystemA は 1 人の開発者によって開発され，そのリリース間隔は 1 か月であった．利用者はシステム試用後に要求変更を提出し，再び 1 か月後にリリースされたシステムを評価した．したがって，開発は漸進的に進められたと言うことができる．進化のためのデータは，利用者へ提供された版を対象に集計した．

2. SystemB: 入金消し込みシステム

   SystemB は 1 人の開発者によって開発され，そのリリース間隔は 2 か月であった．開発者は，数週間の試用期間後に利用者から要求変更を受け取り，次の開発を進めた．したがって，SystemB も漸進的に開発されたとみなすことができる．

3. SystemC: 証券管理システム

   SystemC は 4 人の開発者によって開発された社内開発システムである．開発途中の要求変更はなかったので，このシステムの開発形態は落水型と言うことができる．

以上の 3 システムはいずれも VisualSmalltalk を用いて開発されていた．次に，システムの開発状況を詳しく説明する．

1.5.1 SystemA

SystemA は全部で 6 版の計測対象を得ることができ，第 0 版のクラス数は 10 クラスであり，最後の第 5 版では 52 クラスであった．8 か月の全体の開発期間中に行われた開発内容は次のとおりである．

第 0 版 実現可能性評価用プロトタイプ開発

第 0 版はプロトタイプとしてシミュレーションシステム構築に対する VisualSmalltalk の実現可能性を検討するために開発された．このプロトタイプは使い捨て型のプロトタイプであり，以降のシステムとの関連は少ない．

第 1 版 基本構成システム構築

シミュレーションを行うために必要な最小限の機能と設備に該当するオブジェクトが開発された．

第 2 版 熱交換アルゴリズムの複雑化

熱交換のためにいくつかのアルゴリズムが提供され，シミュレーション設備の種類が増やされた．

第 3 版 シミュレーション設備の多様化とシミュレーション方式の変更

さらに設備の種類が増加し，熱交換のための冷却媒体の流れを制御する機構が追加された．同時に，シミュレーション設備を構成するエディタの操作性向上が要求された．

第 4 版 シミュレーション設備の追加と整理

新しい設備が追加されると同時に不要な設備を削除するように要求変更があった．

第 5 版 操作性改善

操作上の不具合の改善が要求された．

オブジェクトの進化過程を観測するにあたり，第 1 版から第 4 版までを観測対象として選択した．第 0 版と第 5 版を観測対象から外したのは，第 0 版が使い捨て型のプロトタイプであったためであり，第 5 版は第 4 版と計測結果に違いが認められなかったためである．

SystemA に各メトリクスを適用して得られた計測結果を，表 1.1 および図 1.1 に示す．図 1.1 の横軸はシステムが利用者に公開された版を表し，縦軸は第 1 版を 1 としたときの各版におけるそれぞれの計測値の相対度数を表している．SystemA の 4 つの版のうち，第 4 版のシステムに定義さたクラス数，メソッド数，総行数はそれぞれ 52 クラス，927 メソッド，8677 行で，第 1 版の 3.25 倍，4.80 倍，5.56 倍となった．図 1.1 は，SystemA が S 字の曲線を描きながら規模を増加させていたことを示している．第 2 版から第 3 版へ至る期間が S 字の顕著な増加傾向を示す期間である．

SystemA では，第 1 版から第 2 版へ至る過程で，シミュレーションの設備を表示する機能を持っていたオブジェクトからシミュレーションの計算を行うオブジェクトを独立させることによって，それまで 2 つの主なクラス継承木によって実現されていたシステムのフレームワークは，3 つのクラス継承木が構成するフレームワークに変更された．NCT（クラス継承木の数）の増加は，このような設計変更を表している．

表 1.1: SystemA の成長

メトリクス	第 1 版	第 2 版	第 3 版	第 4 版
NCD	16	26	47	52
相対度数	1.00	1.63	2.94	3.25
SLOC	1560	3714	8044	8677
相対度数	1.00	2.38	5.16	5.56
SNOM	193	436	885	927
相対度数	1.00	2.26	4.59	4.80
NMN	115	241	452	463
相対度数	1.00	2.10	3.93	4.03
NCT	5	6	6	6
相対度数	1.00	1.20	1.20	1.20

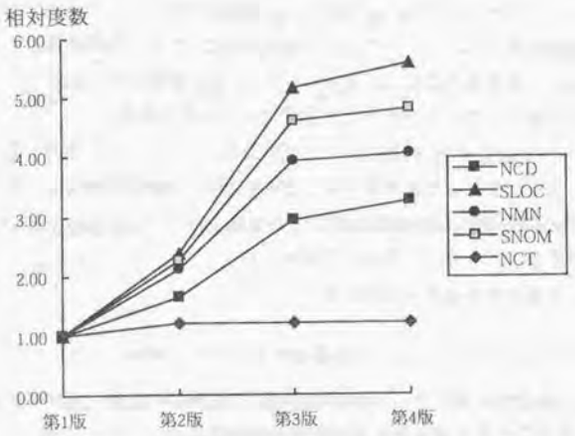


図 1.1: SystemA の定量的開発傾向の推移

1.5.2 SystemB

システムBの成長経過を表1.2および図1.2に示した．図1.2は，SystemAと同様，第1版におけるそれぞれのメトリクスで計測した結果を1とし，以降の版の計測値の倍率として相対度数を求めて縦軸に表したグラフである．SystemBの4つの版のうち，第4版のシステムに定義されていたクラス数，メソッド数，総行数はそれぞれ62クラス，2644メソッド，20470行で，第1版の1.68倍，3.07倍，3.51倍となった．図1.2がSystemAのようなS字の成長曲線を描いていないのは，SystemBとSystemAのシステムの性格の違いに由来するものである．SystemAは，シミュレーションシステムであり，システムの将来像を開発者が予想できない状況で漸進的に開発が進んだが，SysmteBの場合はビジネス系の定形処理であったため，開発の初期にシステムの将来像を開発者が想定できた．開発者へのインタビューから，第1版でフレームワークの妥当性を検証できたため，システム全体の構造は第4版へ至るまでに変更する必要なかったことも確認した．そのような状況下でも，開発期間中クラスの再構成は行われていた．これは，クラス数（NCD）が第2版以降減少している点から読みとることができる．システムの将来像が想定できいた場合でもクラスの再構成が起きていたということは，開発当初の設計が，利用者要求の変更や追加によって変更されたことを意味しており，利用者環境の変化がクラスを進化させる原動力となっていたことを示すものと解釈できる．

SystemBのもう一つの特徴として，利用者インタフェースを実現するクラスが2クラスだけであった点を挙げることができる．必要な利用者インタフェースは，VisualSmalltalkの開発環境として提供されているPARTS Workbenchと呼ばれるコンポーネントを用いて開発された．

次に，全体の開発過程を説明する．

第1版 システムのフレームワーク評価用プロトタイプ構築

この版はフレームワークの評価を兼ねて開発された進化型プロトタイプであり，データベースと接続する前に顧客へリリースされた．

第2版 フレームワークに基づいた試用版システム構築

システムの試用向け機能が第1版のフレームワークを基に構築され，利用者インタフェースの見直しも同時に行われた．

表1.2: SystemBの成長

メトリクス	第1版	第2版	第3版	第4版
NCD	37	81	71	62
相対度数	1.00	2.19	1.92	1.68
SLOC	5839	18677	20579	20470
相対度数	1.00	3.20	3.52	3.51
SNOM	861	2491	2705	2644
相対度数	1.00	2.89	3.14	3.07
NMN	642	1762	1944	1928
相対度数	1.00	2.74	3.03	3.00

第3版 システム運用版の開発

利用者インタフェースの追加要求を受け取り，システムの運用版の仕様が決定され開発され，データベースと接続した版が利用者へリリースされた．ここで要求された利用者要求変更件数は59件で，そのうち70%が利用者インタフェースの変更要求であった．他の30%は機能の追加，変更，削除である．主な機能追加は，画面に表示された一覧表の操作に関するものである．

第4版 システムの使い勝手の改善

利用者要求変更は6件で利用者インタフェースと機能変更要求とが50%づつであった．

1.5.3 SystemC

SystemCで観測された落水型の開発過程を図1.3，その値を表1.3に示す．図1.3は第00版の各メトリクスを適用して求めた計測値を1としたとき，それぞれの版で求めた値を第00版に対する倍率を相対度数として縦軸にとり，システムの成長グラフを描いた．全体の成長が約1か月でほぼ収束しているのは，その後の開発で要求変更や仕様の変更が起こらなかったことを表している．

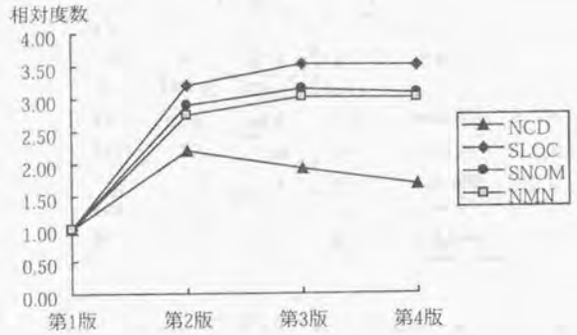


図 1.2: SystemB の定量的開発傾向の推移

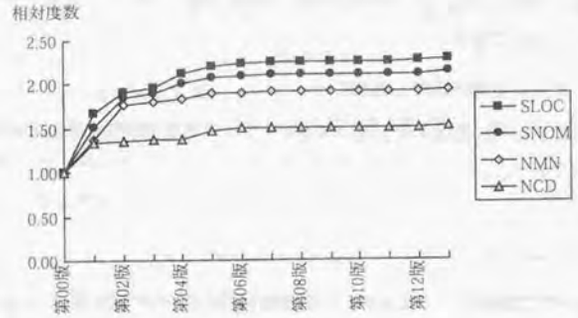


図 1.3: SystemC の定量的開発傾向の推移

表 1.3: SystemC の成長

メトリクス	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
NCD	88	118	119	120	120	129	132
相対度数	1.00	1.34	1.35	1.36	1.36	1.47	1.50
SLOC	6570	10966	12578	12846	13864	14405	14624
相対度数	1.00	1.67	1.91	1.96	2.11	2.19	2.23
SNOM	698	1060	1294	1319	1406	1449	1457
相対度数	1.00	1.52	1.85	1.89	2.01	2.08	2.09
NMN	510	695	904	914	933	964	968
相対度数	1.00	1.36	1.77	1.79	1.83	1.89	1.90

メトリクス	第 07 版	第 08 版	第 09 版	第 10 版	第 11 版	第 12 版	第 13 版
NCD	132	132	132	132	132	132	133
相対度数	1.50	1.50	1.50	1.50	1.50	1.50	1.51
SLOC	14738	14731	14735	14736	14737	14814	14934
相対度数	2.24	2.24	2.24	2.24	2.24	2.25	2.27
SNOM	1465	1466	1466	1466	1466	1466	1487
相対度数	2.10	2.10	2.10	2.10	2.10	2.10	2.13
NMN	976	976	975	975	976	976	981
相対度数	1.91	1.91	1.91	1.91	1.91	1.91	1.92

第 13 版のシステムの総クラス数，メソッド数，行数はそれぞれ 133 クラス，1487 メソッド，14934 行で，第 00 版から 1.51 倍，2.13 倍，2.27 倍となっていた．利用者インタフェースは，SystemB と同様，PARTS Workbench を用いて開発されており，5 クラスだけがインタフェースに使われる部品として定義されていた．

第 2 章

システムレベルから観測した進化過程

2.1 概要

本章では，オブジェクトの進化過程を利用者環境変化および技術的環境変化と関連づけた分析結果について議論する．

システムレベルの分析では，クラスの行数やメソッド数，メソッドの行数といった計測値の分布が，左に最頻値を持ち右に尾を引く形状を持っており，システム進化によらず，その形状は大きく変化しないことが明らかとなった．また，この分布には，例外的な規模を持ち，他のオブジェクトに比べて早く進化するオブジェクトも存在していた．このようなオブジェクトは，開発者が問題箇所として注目し，設計変更の対象として抽出していたオブジェクトと一致した．以上の調査から，計測値の分布から特異な進化を示すオブジェクトを発見すれば，設計変更候補のオブジェクトを抽出できることが明らかとなった．

さらに，複数のクラスが連動して進化する過程を，開発者の設計意図と関連づけた．本章では，これらのオブジェクトの計測値について，頻度分布の安定性を議論したあと，進化過程と設計変更事由および設計変更による効果を対応づけながら議論する．

2.2 計測値の頻度分布の安定性について

図 2.1に SystemA に定義されたメソッドの行数 (MLOC) について，その頻度分布と累積度数分布をシステムの版ごとに示し，表 2.1にその基本統計量を

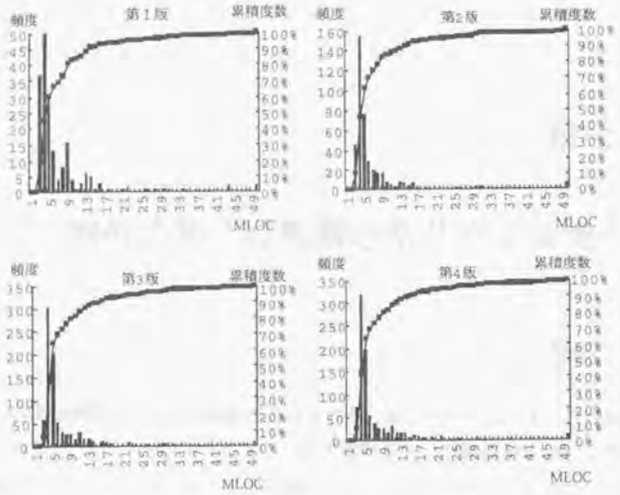


図 2.1: SystemA のメソッド行数 (MLOC) の頻度分布と累積度数分布

示す．SystemA の各版の分布を図 2.1の左上，右上，左下，右下にそれぞれ示した．

各版を比較すると，中央値には変化が起きていないことがわかる．本研究で調査を行った 3 システムの開発言語である Smalltalk は，メソッドを定義するためのテンプレートを提供している，テンプレートは，メッセージ名，コメント行，空行，メソッド本体の 1 行からなる．SystemA の計測結果で MLOC の最小値が 3 となっているメソッドは，本体が 1 行だけのメソッドである．また，Smalltalk では，プログラマがメソッドを定義する際に SystemBrowser と呼ばれるエディタを使う．このエディタを使うと，平均的なメソッド行数はウィンドウ内で一瞥できる 10 行程度の長さになると予想される．しかし，表 2.1に示すように，メソッドの行数の最大値が，数百行となるものもあった．また，進化過程の調査から，このメソッドは版を重ねるたびに行数を増大させていることもわかった．

表 2.1: SystemA のメソッド行数の基本統計量

	第 1 版	第 2 版	第 3 版	第 4 版
平均値	8.08	8.52	9.09	9.36
中央値	5	5	5	5
標準偏差	10.79	16.01	19.53	21.52
最小値	3	3	3	3
最大値	116	261	410	427
メソッド数	193	436	885	927

表 2.2: SystemB のメソッド行数の基本統計量

	第 1 版	第 2 版	第 3 版	第 4 版
平均値	6.78	7.50	7.61	7.74
中央値	4	4	4	5
標準偏差	8.55	8.22	7.93	7.76
最小値	1	1	1	1
最大値	99	99	99	99
メソッド数	861	2491	2705	2644

SystemB および SystemC におけるメソッド行数の基本統計量を表 2.2および表 2.3，表 2.4に，その頻度分布を図 2.2および図 2.3に示す．いずれの例も，SystemA と同様，中央値には大きな変化はなく，分布の形状も安定している．

これらの分布の形状は Lorenz らが計測した結果ともよく似ている [37]．

ここで，これらの分布に統計モデルをあてはめることを試み，分布の形状が表す意味について考えることにする．適用を試みた統計モデルは幾何分布である．幾何分布の確率 $f(x)$ は，成功確率 p とし，最初の成功 Sが出現するまでのベルヌーイ試行回数 x を確率変数とするとき，$x = 1, 2, 3, ...$で，

$$f(x) = pq^{x-1}, x = 1, 2, 3, ...$$

で与えられる．ただし，失敗確率 $q = 1 - p$ とする．幾何分布をメソッド行数

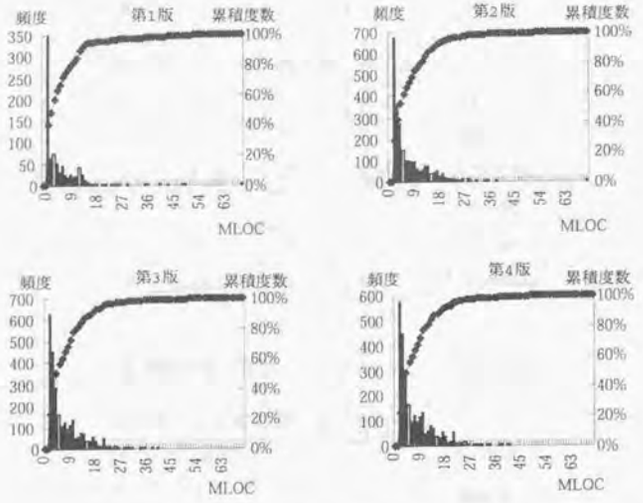


図 2.2: SystemB のメソッド行数 (MLOC) の頻度分布と累積度数分布

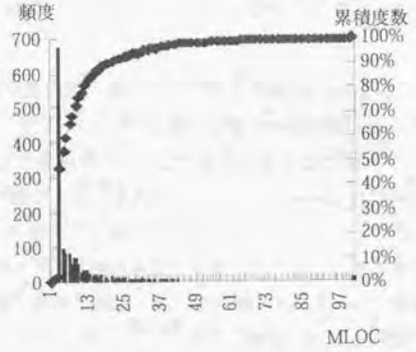


図 2.3: SystemC のメソッド行数 (MLOC) の頻度分布と累積度数分布

表 2.3: SystemC のメソッド行数の基本統計量

	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
平均値	9.41	10.35	9.72	9.74	9.86	9.94	10.04
中央値	4	5	5	4	4	5	5
標準偏差	12.90	14.46	13.73	13.76	13.56	13.65	13.71
最小値	2	2	2	2	2	2	2
最大値	168	171	177	177	178	178	178
メソッド数	698	1060	1294	1319	1406	1449	1457

表 2.4: SystemC のメソッド行数の基本統計量 (つづき)

	第 07 版	第 08 版	第 09 版	第 10 版	第 11 版	第 12 版	第 13 版
平均値	10.06	10.05	10.05	10.05	10.05	10.11	10.04
中央値	5	5	5	5	5	5	5
標準偏差	13.69	13.67	13.68	13.68	13.68	13.69	13.62
最小値	2	2	2	2	2	2	2
最大値	178	178	178	178	178	178	178
メソッド数	1465	1466	1466	1466	1466	1466	1487

表 2.5: 3 システム全体で定義されたメソッド行数の基本統計量

平均値	標準偏差	最小値	最大値	メソッド数
8.15	11.79	1	427	12627

の頻度分布にあてはめると「メソッドの行を先頭から順に検査したとき，最初にメソッドの終端が出現するまでの行数 x の確率がメソッド終了確率 $p = 1/($メソッドの平均行数$)$ の幾何分布に従う」と解釈できる．図 2.4に本研究の対象 3 システムで定義された全メソッドの行数を標本とする頻度分布と累積度数分布を表わした．表 2.5にこの標本の基本統計量を示す．頻度分布の階級は，メソッド行数の階級を 2 行ごとに区切って表した．図の折れ線グラフは，標本の平均値をパラメータとする幾何分布から頻度分布を求めて，理論値として表わたものである．

直観的には，メソッド行数の分布が理論的に求めた幾何分布によく従っているように見える．

メソッド行数の分布が幾何分布に従っていることを統計的に検定するために，5%の有意水準を用いてピアソンの適合度検定を行った．ここで提起する帰無仮説 H_0 と対立仮説 H_1 は次のようになる．

$$H_0: p_1 = p_{1_0}, p_2 = p_{2_0}, p_3 = p_{3_0}, \ldots, p_k = p_{k_0}$$

$$H_1: \text{少なくとも 1 つの } p_i \neq p_{i_0}$$

ただし，階級 i の観測確率を p_i とし，理論確率を p_{i_0} とした．ピアソンの適合度検定では，n 回の試行（この場合は全メソッド数=12627）において，以下の式を用いた検定を行う．

$$\text{適合度検定}: \chi^2 = \sum_{i=1}^{k} \frac{(np_i - np_{i0})^2}{np_{i0}} \sim \chi^2(k-2)$$

上記の χ^2 の値が k を階級の数とした $\chi^2(k-2)$ の値よりも小さければ帰無仮説を採択し，大きければ帰無仮説を棄却する．検定の結果，4 行以下の観測頻度と理論値との差が大きく帰無仮説を棄却せざるを得なかった．観測結果から求めた頻度分布が幾何分布に従わない要因として，

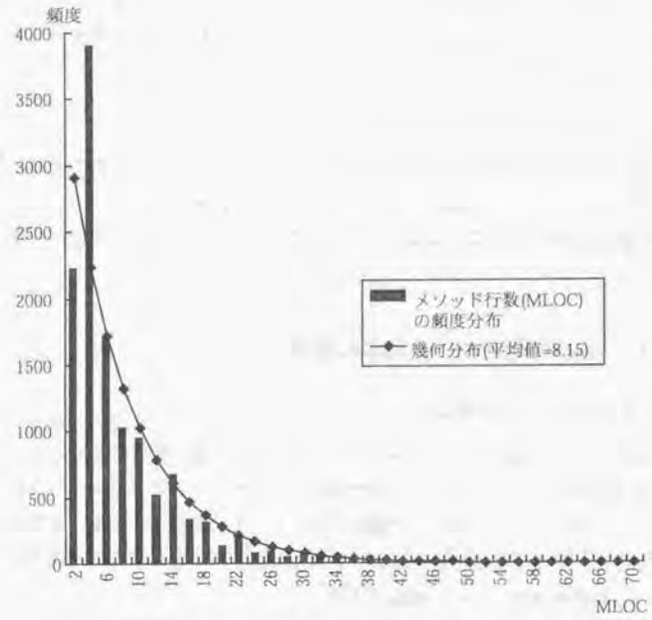


図 2.4: 3 システム全体で定義された全 12627 メソッドを対象としたメソッド行数 (MLOC) の頻度分布と幾何分布による理論分布の比較．幾何分布のパラメータは，観測データから求めた平均値=8.15 を用いた．

- 行数の小さい部分では，1行の本体のメソッドでも，空行の有無，コメント行の有無などのプログラマの嗜好が分布に大きな影響を与える．
- 検定で用いた幾何分布は，すべてのメソッドがx行で終了する確率を等しいと仮定する．しかし，ここで集計したメソッドの中には，仕様上の複雑度が異なるメソッドも含まれており，仕様の記述が終了するために要する行数の平均値は，全メソッドで一定となるとは考えられない．これは，すべての行における終了確率が一定であるとする幾何分布の前提条件を満足しない．

といった要因を考えることができる．

検定の結果，帰無仮説は棄却されたが，メソッドの行数の分布が幾何分布に近い分布となることを確認することはできた．この研究成果は，メソッドの平均行数を見積もれれば，規模の大きいメソッドの割合を推定できることを示唆している．

2.3 技術的環境変化と進化の関係

2.3.1 SystemA の進化

新たなクラス継承木の出現やクラスの分割は，将来の開発のためにクラスの仕様を整理するために行われる設計変更であるから，システムレベルにおけるオブジェクトの技術的環境への適応と考えることができる．また，クラスの名称変更も，利用者環境の変化によって開発者自身が問題領域への理解を深めるという技術的環境変化への適応である．

図2.1では，分布の右端に現われる少数のメソッドを観測した．これらのメソッドは開発当初から定義され，第4版までの全期間にわたって存在し続けたメソッドである．行数の変化を追跡調査した結果，版を増すごとに，これらのメソッドはその行数を増加させていることがわかった．開発者は，このようなメソッドの存在について次のように述べている．

「巨大なメソッドが，システムの成長に伴って規模を増大させたのは，偶然ではなく，要求変更をこれらのクラスに集中させてシステム全体に及ぶ設計変更を避けたためである．メソッドの規模が増大したのはフレームワークに原因

があった．フレームワークの設計上の問題には第3版の要求変更を受け取った時点で気付いたが，システム全体のフレームワークを変更する時間的な余裕はなかったため，小数のメソッドで要求変更に対応する設計方針を選択した．」

第3版では，設備の種類の増加と，熱交換のための冷却媒体の流れを制御する機構の変更が要求された．巨大なメソッドを保持していたクラスは，様々な設備との熱交換条件を保持し，設備の配置に従って熱交換を制御する部品である．第3版で，このクラスに冷却媒体の流量などを制御するための条件を追加している．同じ要求追加に対しては，流量を制御するための役割分担を他の設備オブジェクトにも与えて対処する選択肢もあったが，開発期間の制約によって，変更内容を一箇所に集中させる戦略が選択されていた．

以上の観測とインタビュー結果から，定量的に特異な進化を示すメソッドには，設計上の問題がある可能性の高いことが明らかとなった．

定量的に特異な進化をするクラスについて，開発者はどのような設計意思決定を行っているのか．SystemA で行われた設計変更を，クラス継承木の進化から観測した．

SystemA は，次に示す3つの主なクラス継承木から構成されている．

- 編集クラス木：シミュレーションに必要な初期値を与える利用者インタフェースのクラス群．
- 表示クラス木；シミュレーションの設備を表わすアイコンなど，設備を視覚的に組み立てる利用者インタフェースを提供するクラス群．設備の組み立てに関する制約条件も保持する．
- 計算クラス木：シミュレーションの計算エンジンを提供するクラス群．

表2.6にはクラス継承木を構成するクラスの数(NCBT)の変化の軌跡を示す．第2版では，計算クラス木が新たに追加されている．第2版の要求追加では，熱交換制御のためにいくつかの熱交換アルゴリズムが提供された．そのため，開発者は，将来の熱交換アルゴリズムの複雑化に対処するために表示クラス木のクラスから熱交換の計算部分だけを取り出して新たなクラスを構築した．こうして形成されたクラス継承木が計算クラス木である．第2版以降，個々のク

表 2.6: クラス継承木を構成するクラスの数 (NCBT) から観測したクラス継承木の進化過程

クラス木名	第1版	第2版	第3版	第4版
編集クラス木	3	5	10	11
表示クラス木	6	8	16	18
計算クラス木	–	6	14	16
物理定数保持クラス木	5	5	5	5
合計	16	26	47	52

ラス継承木には新たな子継承木やクラスも追加されているが，新たなクラス継承木は追加されていない．

それでは，SystemA における技術的環境の変化と，定量的なクラスの進化過程との関係について分析を進めよう．図 2.5にクラス継承木の構造が変化した様子を示す．

この図には，次の 2 つの群に属するクラスが表示されている．

- 第 1 群：クラスの分割に関わったクラス群（第 1 版から第 2 版の間に起こった表示クラス木の分割）
- 第 2 群：クラス継承木の中でスーパークラスの組み替えに関わったクラス群

図 2.6に SystemA のクラスのメソッド数から観測した変化の軌跡を示す．図で示された折れ線は，それぞれクラスの進化の軌跡を表す．グラフの横軸はシステムの版番号を表し，縦軸はメソッド数を表す．

1. 第 1 群のオブジェクトの進化：クラス分割

   SystemA の第 1 版で最大の規模（行数とメソッド数）を持っていたクラスは表示クラス木に属するクラス P5 である．開発者は，第 2 版の要求追加によって，表示クラス木を構成していたすべてのクラスを分割し，熱交換の計算を実行する計算クラス木を新たに定義した．このような設計

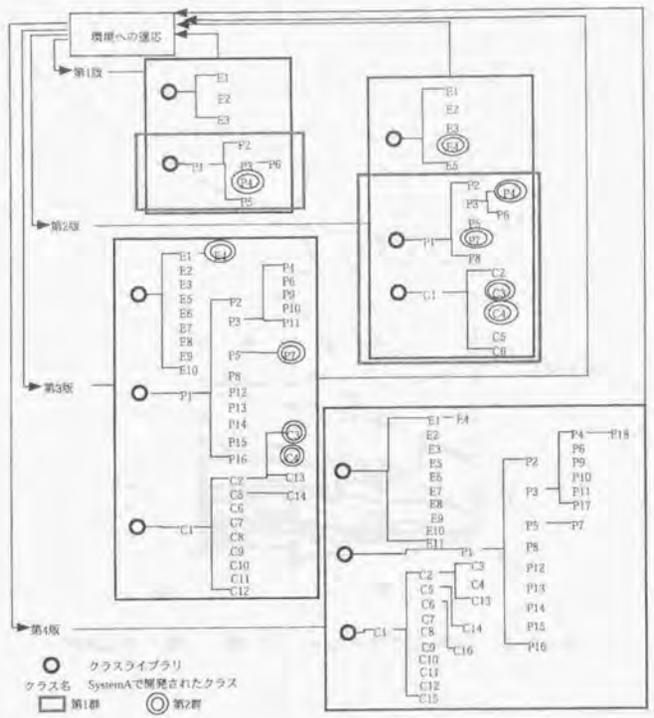


図 2.5: SystemA のクラス継承木の進化過程（E_i:編集クラス木，P_i:表示クラス木，C_i:計算クラス木）

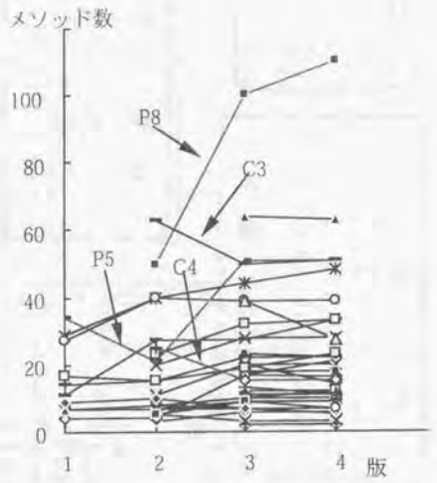


図 2.6: SystemA のクラスのメソッド数から観た進化過程

変更を行った結果，最大の規模を持っていたクラス P5 は他のクラスと同程度の規模となった．

2. 第 2 群のオブジェクトの進化：継承構造の再構成

第 2 群に属するクラスには，第 2 版で，最大の規模を持つ計算クラス木に属するクラス C3 が含まれている．図 2.7に，SystemA の第 3 群で観測された継承構造の再構成過程を示し，各クラスのメソッド数を括弧内に示した．SystemA のクラス C3 は第 2 版から第 3 版に至る過程でメソッド数を 63 から 50 に減らし，この間に，クラス C3 のスーパークラスは，第 2 版でスーパークラスを共有していたクラス C2 へ変更された．クラス C2 は第 2 版でクラス C4 ともスーパークラスを共有していたが，第 3 版ではクラス C4 のスーパークラスとなった．図 2.6に見られるように，クラス C4 の進化過程でも，第 2 版から第 3 版に至る過程でメソッド数を 25 から 15 に減らすといったクラス C3 と同様の変化を観測することができる．第 2 版から第 3 版に至る過程の継承構造の変更は，これら 3 クラスに着目した設計変更であり，クラス C3，C4 の規模を縮小させる効果をもたらしていた．

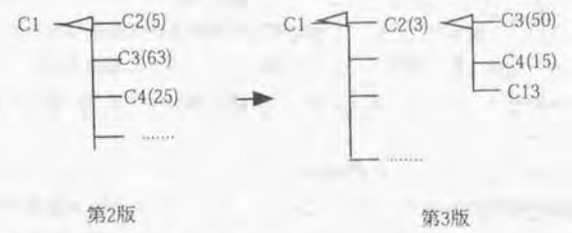


図 2.7: SystemA における継承構造の再構築（クラス名とそのメソッド数（括弧内），△は継承を表す）

第 1 群，および第 2 群で設計変更対象となっていたクラスは，いずれも設計変更時の版の最大メソッド数を持っていたクラスであったことから，開発者が規模の大きいクラスを設計変更対象として注目していたと解釈

できる．ただし，第 3 版で最大規模となっていたクラス P8 に対する設計変更が行われなかったのは，利用者環境の変化が収束しており，再設計を行うコストと，再設計によって向上する拡張性の改善との均衡が成り立たなくなったためと考えられる．

3. 名称変更

SystemA では第 2 版から第 3 版に至る過程で，9 クラスがその名称を変更している．これは第 3 版の全クラスの 1/4 に該当する．名称変更は，熱交換シミュレーションの設備の多様化によって，新たに追加された類似設備に対応するクラスと既存クラスとを区別するために行われていた．

漸進型開発では，利用者環境の変化によって徐々に開発者の問題領域への理解が深まり，ここで観測したように，1/4 のクラスの名称が変更される場合もある．この例から，開発者にとって，そのクラスの役割の識別子として使われているクラス名称は，開発の全期間で変化しないとは言えないことが確認できた．

2.3.2 SystemB の進化

SystemB では，表 2.2に示したように，最大規模を持つメソッドは安定していた．これは，規模が大きいことが必ずしも設計上の問題を持ったオブジェクトであるとは限らない例である．したがって，設計上の問題点は，ある時点における規模だけではなく，オブジェクトの進化過程から発見しなければならないと言える．

SystemB では，SystemA で行われたようなシステムのフレームワークを変更する設計変更は起きていない．しかし，図 2.8に示すクラス継承木の構造の変化では，クラスの統合，クラスの削除，継承木の再構成，継承木の消滅といったオブジェクトの技術的環境への適応を観測できた．

1. クラス統合

SystemB の第 1 版では，入金消し込みの顧客に関するデータは，顧客の認証を行う画面操作を行うためのオブジェクト，顧客を認証するために必要な情報を保持するオブジェクト，その他の顧客登録情報を保持する

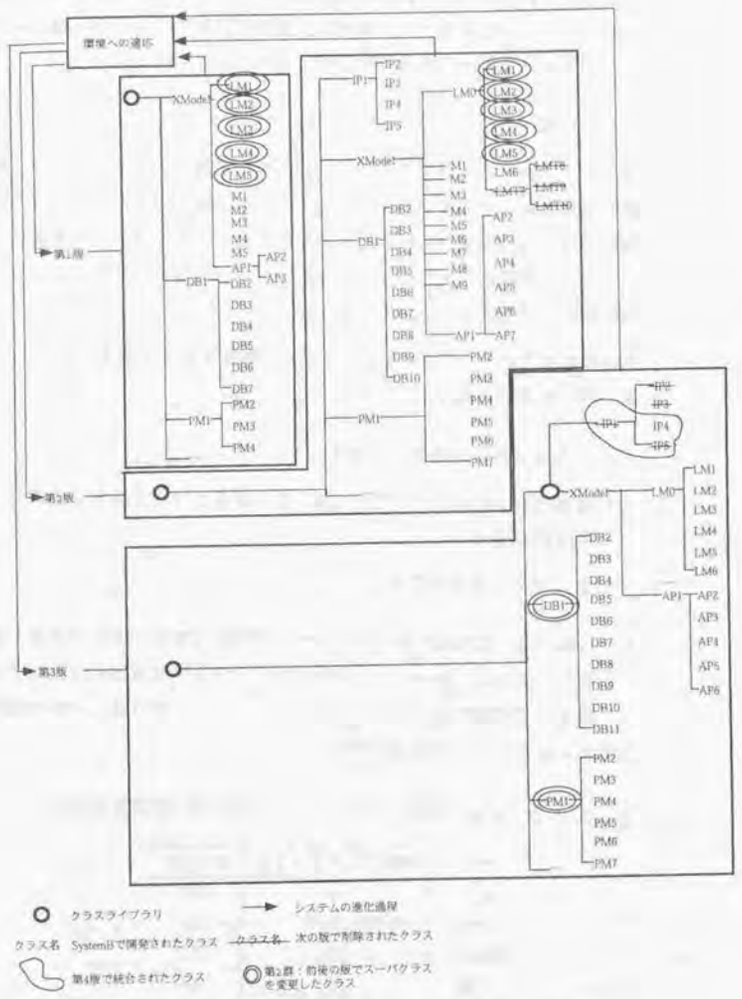


図 2.8: SystemB のクラス継承木の進化過程

オブジェクトの 3 つに分割されて管理されていた．しかし，第 2 版では，これらのクラスは統合されて新たなクラスが定義された．この設計変更は，クラスの凝集度を向上させる効果がある．

2. クラス削除

表 2.7は SystemB におけるクラス削除に関する開発者へのアンケート調査結果をに示した表である．この表のクラス分割の項目には，クラス分割によって名前変更が行われたものも含まれている．表から，不要によるクラスの削除は全体の 25%にすぎず，クラスの分割／統合によるクラス削除が 75%を占めていることがわかる．

名前が変更された 13 クラスについて，名称変更理由を調査したところ，次の項目が挙げられた．

- 日本語名称を英語名称に変更
- 類似クラスの混乱をさけるため，より詳細な仕様を表現できるクラス名称に変更
- より適切な名前を発見．

第一の原因は非英語圏の開発者に発生する技術的環境の変化である．第二と第三の理由は，SystemA で観測したクラス名称変更理由と同様であり，開発者が問題領域を理解するという，クラスを取り巻く技術的環境の変化によるクラスの進化と解釈できる．

表 2.7: クラス削除の理由とその割合（() 内はメソッド数を表す）

理由	削除されたクラス数	割合 (%)
統合	9	20.9
分割	10	23.3
名称変更	13	30.2
不要	11	25.6
合計	43	100.0

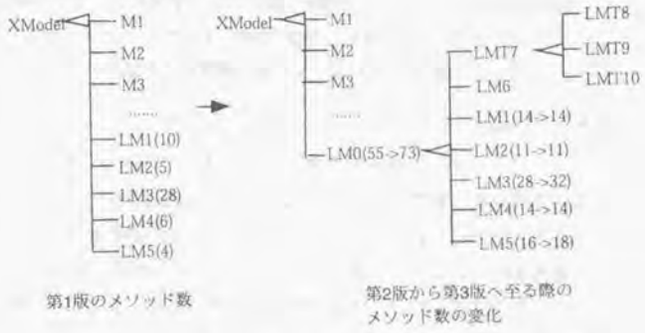


図 2.9: SystemB における継承構造の再構築

3. 継承木の再構築

図 2.9には，SystemB における継承構造の再構築の例を示す．第 1 版のクラスの横に表示した数字は第 1 版のクラスのメソッド数を表している．第 2 版と第 3 版のメソッド数は矢印の前後に示した．これらのクラスは図 2.8に示したように，SystemA で観測されたクラス進化の第 2 群に属するクラスである．クラス XModel は第 1 版で 11 のサブクラスを持っており，第 2 版では，新しいクラス LM0 を継承するクラスとして，このうちの 5 クラスが選択されて継承構造の再構築が行われた．これらのクラスは，一覧表で表示された入金情報や顧客情報の多重選択や選択対象に対する情報の絞り込みといった操作に関する役割を分担するクラスであり，利用者インタフェースの追加要求を反映させるための設計変更の対象となったクラスである．

設計変更の方針にはいくつかの選択肢が考えられるが，ここでは，新たな機能が要求されたクラス群に対して新しいスーパークラスを定義し，複数のクラスに追加すべき機能をこのスーパークラスに吸収させ，サブクラスの進化を緩やかにする効果を得ている．第 2 版から第 3 版に至る期間は，利用者インタフェースの追加要求とともに，システムの最終版

の仕様が決定され開発された期間である．この期間に，クラスLM0は，そのメソッド数を55から73に増加させているが，クラスLM0のサブクラスのメソッド数は，クラスLM0ほど増加していない．

4. 継承木の消滅

図2.8のクラスLMTの継承木は第2版から第3版の期間で消滅した．この開発期間では，構築したシステムと別会社で開発されたデータベースとの結合が行われており，消滅したLMTのクラス継承木はデータベースが提供されていなかった第2版でデータベースを擬似的にテストするためのクラスであった．このような継承木の消滅は，システムの開発時に予想されるものであり，予測できない環境変化に適応する進化というよりもシステムの成長過程の1事象と捉えられる．

継承木の消滅は，その他に，直線的な継承構造を避けるための設計変更でも起こる．漸進的な開発が進められる過程では，一度定義した継承構造がクラスの削除などにより直線的な継承構造や無意味な継承構造となってしまうこともある．図2.8のクラスIP1の継承木では，第4版の継承木消滅の過程で，残されたサブクラスとスーパークラスとの統合が同時に起きている．このクラス継承木に属する5クラスのうち，再利用したクラスがIP1,IP2,IP3,IP4の4クラスあったが，第4版を開発する時点で，再利用クラスが提供していた機能を他のクラスのメソッドとして取り込むことによって，IP2とIP3の2クラスが不要となった．また，残されたSystemBのために開発されたクラスIP5の機能は，継承木内に残っていた再利用クラスIP4に取り込まれ統合された．その結果，これらのスーパークラスであったクラスIP1の抽象化の意味がなくなり，この継承木は消滅した．ここでは，再利用クラスの見直し，再利用クラスの進化，継承構造の見直しによる再利用クラスの再構成が行われていたことになる．いずれの設計変更も利用者へ提供する機能は変化しないため，技術的環境への適応と考えることができる．

機能追加は利用者環境の変化である．しかし，新しい利用者環境にクラスが適応しようとしたとき，クラス構造やシステムの構造が複雑になり，可読性や将来の生産性が低下することはある．そこで，技術的環境に適応する必要が生じる．設計変更は，このような問題を解決するために経験的に選択されてきたが[56]，その妥当性は，ここで調査したように，設計変更後のクラスの進化過程によって評価することができる．

2.4 考察

Lorenzらは，平均値から大きく外れた値は不適切な設計であることを表わしていると言っているが[37]，本研究によって，メソッドの規模だけから設計上の問題を指摘できないことが明らかとなった．たとえば，複雑な計算を行うメソッドの行数は，メソッドの平均行数に比べて大きくなるが，このようなメソッドを分割する必要はない．問題なのは，大規模メソッドが要求変更の影響を受けて増大化傾向を続けて進化することである．規模が大きく，変更が多いということは，バグを生む原因も高まると考えられる．したがって，オブジェクトの変化を時系列で観測し，進化過程の特徴から特異な進化を示すオブジェクトを発見することには意味がある．

進化過程の調査から設計上の問題点を発見できるのは，計測値の分布がシステムの成長によらずに変化しないという一般的に観測された傾向を根拠にしている．平均値や中央値が大きく変化しないオブジェクトは，メソッド行数の他に，クラスのメソッド数，行数がある．付録Bに，各システムのメソッド行数，クラスのメソッド数，クラスの行数の各頻度分布と，幾何分布より求めた理論的な分布とを重ねたグラフを掲載した．いずれの分布も，理論値から求めた頻度分布によく従う．頻度分布の特徴から，将来開発されるクラスやメソッドの規模と数を見積もることが可能であると考える．しかし，頻度分布を見積もりに応用するためには，さらに他のシステムについて調査を行う必要があるだろう．

第 3 章

クラス継承木の進化的特徴

3.1 概要

クラス継承木の進化過程について SystemA を調査した結果，クラス継承木には 1 メソッド当たりの行数の値を保持し続ける傾向があることを発見した．この傾向は，たとえば，あるクラスが継承木に追加された場合，そのクラスは継承木が持つ 1 メソッド当たりの行数に近い値をもつように設計されるという意味を持ち，たとえ初期の値が継承木の持つ値に近くなかったとしても，再設計によって，継承木が持つ値に近い値を持つように進化するという意味と持つ．本章では，クラスの 1 メソッド当たりの行数を C 値（Characteristic Value）と呼ぶ．また，クラス継承木の C 値とは，クラス継承木に属するクラスの C 値の平均値を指す．本研究では，クラス継承木の進化的特徴を捉えるために，C 値に関するクラス継承木の進化の仮説を提起し，他の 2 システムを用いて仮説の検証を行った．

仮説を検証した結果，C 値には次の 3 つの特徴があることがわかった．

- C 値はクラス継承木に固有の値である．
- C 値はシステムが成長しても安定しており，クラスは，自分が属するクラス継承木の C 値を持つように進化する傾向がある．
- C 値は開発者に依存せずに決まる．

本章の構成は次のとおりである．第 3.2節では SystemA で調査した結果を説

表 3.1: SystemA におけるクラスの性質を表す計測値の相関分析結果（相関係数）

	CNOM	CLOC	DIT	NOC	NIV
CNOM	1.00				
CLOC	0.88	1.00			
DIT	-0.01	-0.18	1.00		
NOC	0.15	0.01	-0.28	1.00	
NIV	0.86	0.62	0.16	0.15	1.00

明し，仮説を提起する．第 3.3節では，他の 2 システムを用いた仮説の検証結果を示す．最後の節では，仮説が示す設計上の意味について議論する．

3.2 予備調査結果

3.2.1 計測値の相関分析結果

SystemA のクラスレベルで計測した値に対して相関分析を行った結果，表 3.1に示すように，メソッド数 (CNOM) とクラス行数 (CLOC)，CNOM とクラスのインスタンス変数の数 (NIV) の間に強い正の相関があった．

散布図を書いたところ，特にクラスの CNOM に対する CLOC の散布図で両者の間に比例関係を確認することができた．図 3.1には，横軸に CNOM をとり，縦軸に CLOC をとったクラスの散布図を示す．図 3.1の左上は，SystemA の全版の全クラスの値を示した散布図である．他の図には，クラスを継承木ごとに分類して散布図を示した．散布図に示したクラスの継承木は，SystemA を構成する次の主なクラス継承木である．

- 編集クラス木：シミュレーションに必要な初期値を与える利用者インタフェースのクラス群．
- 表示クラス木：シミュレーションの設備を表すアイコンなど，設備を視覚的に組み立てる利用者インタフェースを提供するクラス群．

表 3.2: 継承木ごとの CLOC と CNOM の間の相関係数

分析対象	全クラス	表示クラス木	編集クラス木	計算クラス木
相関係数	0.871	0.969	0.951	0.994

- 計算クラス木：シミュレーションの計算エンジンを提供するクラス群．

ここで，クラスの 1 メソッド当たりの行数の値を C 値 (Characteristic Value) と定義する．すなわち，クラス i について，その行数を $CLOC_i$，メソッド数を $CNOM_i$，C 値を CV_i とすると，次の関係が成り立つ．

$$CLOC_i = CV_i \cdot CNOM_i$$

図 3.1の矢印は第 3 版で新たに定義された編集クラス木のクラスが，第 4 版に至る期間で変化した軌跡を表している．矢印の頭が，継承木の C 値に向かっていることから，この間に行われた設計変更がクラスの C 値を継承木の C 値へ向かわせたと考えることができる．開発者はインタビューのなかで，第 3 版のクラスの設計には違和感があり，第 4 版を開発する際に修正したと言っていた．このような調査結果は，クラスの C 値と継承木の C 値との差が，開発者にとって直観的な設計上の不適切さとして認識されていたことを表している．

表 3.2には，全クラスと継承木ごとに求めた CNOM に対する CLOC の相関係数を示す．各継承木の相関係数が全クラスについて求めた相関係数より大きいことから，C 値をクラス継承木に分割して観測することにした．表 3.3に SystemA の各クラス継承木における C 値の基本統計量を示す．直観的にクラス継承木の C 値は継承木ごとに異なり，システムが成長しても大きな変化がないように見えるが，これを統計的手法を用いて検定する．

検定にあたり帰無仮説 H_0(C値固有) を提起し，2 組ずつ継承木を抽出して C 値の平均値について有意水準 5%の両側 t 検定を行う．

H_0(C値固有) : 2 つのクラス継承木の C 値は等しい

表 3.4に t 検定によって得られた有意水準 P 値を示す．P 値が検定で用いた有意水準 5%よりも小さいことから，帰無仮説 H_0(C値固有) を棄却できる．す

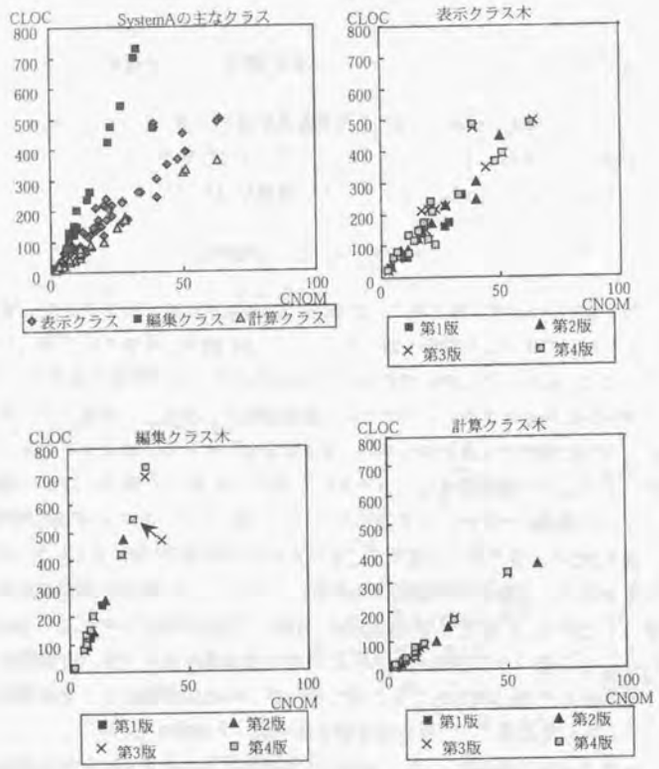


図 3.1: SystemA のクラスのメソッド数 (CNOM) に対するクラス行数 (CLOC) の散布図

表 3.3: SystemA における各継承木の C 値に関する基本統計量

表示クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	7.63	8.44	8.63	8.93
標準偏差	1.89	1.27	1.62	1.78
最小値	5.55	6.10	6.00	6.00
最大値	10.86	10.50	12.29	12.60
クラス数	6	8	16	18
編集クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	15.10	16.34	16.32	16.24
標準偏差	2.16	2.63	3.11	4.07
最小値	13.14	13.14	12.00	7.00
最大値	17.14	20.61	21.81	22.09
クラス数	4	6	11	12
計算クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	-	5.43	5.68	5.74
標準偏差	-	0.70	0.90	0.85
最小値	-	4.40	4.18	4.18
最大値	-	6.33	7.00	7.00
クラス数	-	6	15	15

表 3.4: SystemA における両側 t 検定によって求めた有意水準：P 値（表の有意水準 P 値が検定で用いた有意水準より小さいとき帰無仮説を棄却する）

	編集クラス木	表示クラス木	計算クラス木
編集クラス木	-	-	
表示クラス木	1.05E-15	-	
計算クラス木	2.22E-19	2.04E-16	-

なわち，すべてのクラス継承木の C 値が等しいとは言えないことを統計的に確認できた．

次にクラス継承木の C 値の安定性を評価するために，各版間の平均値と分散に関する帰無仮説 H_0(平均値安定), H_0(分散安定) を提起し，有意水準 5%の両側 t 検定および両側 F 検定を実施した．

H_0(平均値安定)：クラス継承木の C 値の平均値はシステムの成長によって変化しない

H_0(分散安定)：クラス継承木の C 値の分散はシステムの成長によって変化しない

表 3.5に，t 検定および F 検定によって求めた有意水準 P 値を示す．表に示す各 P 値は検定で用いた有意水準 5%より大きいから，帰無仮説を棄却できない．すなわち，システムが成長したとき，各クラス継承木の平均値および分散が変化するとは言えないという結論を統計的に得ることができた．

3.2.2 クラス継承木の進化的特徴に関する仮説

SystemA において観測されたクラス継承木の進化的特徴が一般的に観測されるか否かを検証するために，次に列挙する 3 つの仮説を検証した．

1. C 値固有仮説：クラス継承木の C 値はクラス継承木間で差異はない．
2. C 値安定仮説：クラス継承木の C 値は，システムが成長しても，その平均値，および分散は変化しない．
3. C 値開発者非依存仮説：C 値は，開発者間で有意な差異はない．

これらの仮説を検証するために SystemB および SystemC のクラス継承木の進化的特徴を調査した．

表 3.5: SystemA の各クラス継承木ごとの C 値の平均値および分散に対する両側 t 検定と両側 F 検定より求めた有意水準：P 値

t 検定	1-2 版	2-3 版	3-4 版
表示クラス木	0.229	0.470	0.662
編集クラス木	0.459	0.989	0.963
計算クラス木		0.485	0.923
F 検定	**1-2 版**	**2-3 版**	**3-4 版**
表示クラス木	0.275	0.273	0.795
編集クラス木	0.793	0.746	0.405
計算クラス木	-	0.590	0.771

3.3 仮説の検証

3.3.1 SystemB の検証結果

クラス継承木の概要

次に，SystemB を構成する 4 つの主なクラス継承木を示す．仮説の検証は，これらのクラス継承木を対象として行った．

- 仲介クラス木：利用者インタフェースとデータベース上のオブジェクト間の仲介を行うクラス群．
- 一覧表操作クラス木：一覧表に表示されたオブジェクトの操作を行うクラス群．
- 永続化クラス木:データベース上の永続オブジェクトに対応するクラス群．
- 表抽出クラス木：データベース上のオブジェクトを一覧表に表示するために，データベースからオブジェクトを抽出するクラス群．

仲介クラス木と一覧表操作クラス木は共通のスーパークラスを継承し，永続化クラス木と表抽出クラス木は，最終版でクラス Object の直下に定義された．

仮説の検証

1. C 値固有仮説の検証

図 3.2にすべてのクラスと 4 つのクラス継承木で求めた各クラスの CNOM に対する CLOC の散布図を示す．図 3.2には，クラスの進化の軌跡を矢印で示した．また，上段の散布図に示した直線の傾きは，各クラス継承木の C 値を表す．

相関分析の結果，クラス継承木の相関係数の最大値は一覧表操作クラス木の 0.994，最小値は表抽出クラス木の 0.915 で，いずれのクラス継承木でも CNOM と CLOC の間に強い正の相関関係を観測できた．相関分析の詳細は，付録 Cに示してあるので参照されたい．

表 3.6にクラス継承木ごとの C 値に関する基本統計量を示す．表では，第 1 版から第 2 版の間で各クラス継承木の C 値が変化しているが，これは第 1 版のクラス数が少ないために発生した誤差と考えられる．

表 3.7は 5%の有意水準の両側 t 検定で求めた各クラス継承木の P 値を示したものである．永続化クラス木と表抽出クラス木の間の P 値は 0.05 より大きいが，他のクラス継承木の間で求めた P 値は 0.05 よりも小さい．以上の結果から，帰無仮説 H_0(C値固有) は採択できない．すなわち，すべてのクラス継承木の C 値は等しいとは言えないことが SystemB においても統計的に検証できた．

2. C 値安定仮説の検証

仲介クラス木に定義されている各クラスの CNOM に対する CLOC の値の散布図を，図 3.2の中段左に示す．円内の点を結んでいる矢印は，第 2 版から第 3 版へ至る期間のクラスの進化の軌跡である．この軌跡の矢印の頭がクラス継承木の C 値へ向いているのは，第 3 版でクラスの設計が変更された結果，このクラスの C 値がクラス継承木の C 値に近い値へ変化したことを表している．この現象は SystemA の編集クラス木で観測した現象と同じである．他のクラス継承木の散布図を観察しても，クラスはクラス継承木の C 値に沿って進化しているように見える．

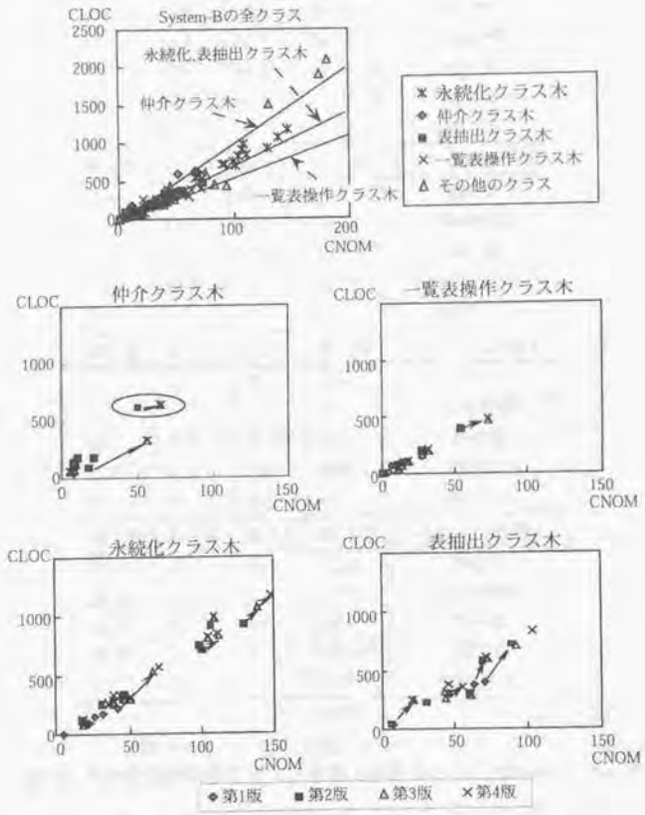


図 3.2: SystemB の CNOM に対する CLOC の散布図

表 3.6: SystemB の各クラス継承木の基本統計量

仲介クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	8.04	12.29	9.06	9.07
標準偏差	1.80	4.53	1.64	1.64
最小値	7.00	4.56	5.74	5.74
最大値	10.11	16.00	10.00	10.00
クラス数	3	7	6	6
一覧表操作クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	8.21	5.28	5.90	5.91
標準偏差	3.64	2.08	0.76	0.76
最小値	5.5	2.00	4.45	4.45
最大値	13.33	7.27	6.66	6.66
クラス数	4	11	7	7
永続化クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	4.97	7.08	7.37	7.49
標準偏差	1.26	0.78	1.10	1.13
最小値	2.25	6.17	5.12	4.95
最大値	6.00	8.58	9.18	9.18
クラス数	8	10	11	11
表抽出クラス木	第 1 版	第 2 版	第 3 版	第 4 版
平均値	5.77	6.80	7.67	7.77
標準偏差	0.30	1.17	2.37	2.38
最小値	5.43	4.85	4.89	4.89
最大値	6.10	8.26	12.29	12.29
クラス数	4	7	7	7

表 3.7: SystemB における両側 t 検定によって求めた有意水準：P 値

クラス木名	仲介	一覧表操作	永続化	表抽出
仲介	-			
一覧表操作	0.000	-		
永続化	0.000	0.004	-	
表抽出	0.001	0.006	0.505	-

表 3.8: SystemB の各クラス継承木で求めた C 値の平均値および分散に対する両側 t 検定と両側 F 検定で求めた有意水準：P 値

t 検定	1-2 版	2-3 版	3-4 版
仲介クラス木	0.164	0.128	0.996
一覧表操作クラス木	0.068	0.459	0.992
永続化クラス木	0.001	0.337	0.812
表抽出クラス木	0.164	0.888	0.861
F 検定	1-2 版	2-3 版	3-4 版
仲介クラス木	0.284	0.041	0.998
一覧表操作クラス木	0.157	0.022	0.991
永続化クラス木	0.196	0.394	0.926
表抽出クラス木	0.144	0.151	0.983

C 値安定仮説を検証するために，クラス継承木ごとに求めたクラスの C 値の平均値および分散に対して，有意水準 5%の両側 t 検定と両側 F 検定を行い帰無仮説 H_0(平均値安定) および H_0(分散安定) を検証した．検定によって求めた有意水準 P 値を表 3.8に示す．ほとんどのクラスについて検定で用いた有意水準 5%よりも検定結果の有意水準 P 値は大きい．永続化クラス木の第 1 版と第 2 版の間で求めた t 検定の有意水準 P 値が 5%よりも小さく，したがって平均値に有意な差異があるという結果が得られたのは，第 1 版の時点でデータベースの設計が終了していなかったため，第 1 版から第 2 版へ至る間に設計変更が起きていたためと考えられる．このような設計変更の例を除くと，帰無仮説 H_0(平均値安定) は棄却できない．すなわち，クラス継承木の版ごとに求めたクラスの C 値の平均値は，隣接する版の間に差異があるとは言えないという結論が得られた．

また，F 検定の結果（表 3.8）では，第 2 版から第 3 版の期間における仲介クラス木と一覧表操作クラス木の P 値は 0.05 よりも小さい．一覧表操作クラス木の P 値は，第 2 版で 2.00，2.50 という分散の値を持つ 2 クラスの C 値が有意な差異を生じさせた原因であった．これらのクラスは，クラス継承木を構成しており，第 3 版では削除された寿命の短いク

ラスであった．そこで，これらのクラスを例外として一覧表操作クラス木から除いたところ，第 2 版から第 3 版の期間における P 値は 0.05 よりも大きくなった．

仲介クラス木は，図 3.2に示したクラスの進化の軌跡でもわかるように，クラスの進化の方向はクラス継承木の C 値に向かって収束している．このように C 値の分散は新たなクラスが追加された際に一時的に大きくなることもあるが，そのような時期を例外と考えると，帰無仮説 H_0（分散安定）を棄却することはできない．

以上の検定結果から，C 値の安定仮説を検証できた．

3.3.2 SystemC の検証結果

クラス継承木の概要

SystemC では，次の 8 つのクラス継承木に着目して，仮説を検証する．ただし，SystemC は漸進的な開発が行われなかったため，C 値安定仮説については検証することができなかった．そのかわり，C 値開発者非依存仮説を検証できた．クラス継承木の担当者と共に，それぞれのクラス継承木の概要を説明する．

- 証券領域クラス木群：証券問題領域のクラスが構成するクラス継承木で，5 つの子継承木，およびその他のクラスからなる．5 つの子継承木のうち 3 つを開発者 a が開発し，残りの 2 つのクラス継承木を開発者 b が開発した．
- 仲介クラス木群：データベース上のオブジェクトと証券領域クラスとの仲介を行うクラスが構成するクラス継承木で，開発者 c が開発した 2 つの子継承木を持つ．
- 定数表クラス木：税金などの定数表を保持するクラスが構成するクラス継承木で，開発者 d が開発した．

仮説の検証

1. C 値固有仮説の検証

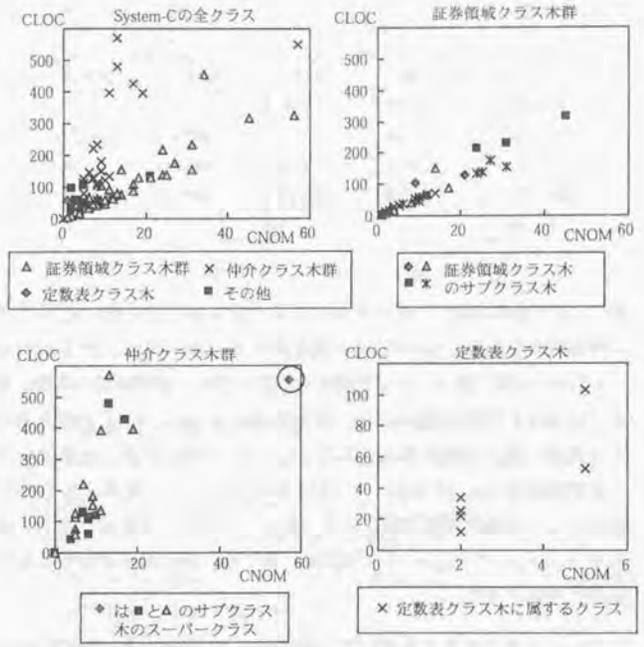


図 3.3: SystemC の CNOM に対する CLOC の散布図

表 3.9: SystemC のクラス継承木における C 値の基本統計量

クラス木名	証券領域 1	証券領域 2	証券領域 3	証券領域 4
平均値	4.87	5.17	5.99	5.58
標準偏差	0.98	1.23	2.54	1.98
最小値	4.00	4.00	4.00	4.00
最大値	6.56	7.50	10.93	9.00
クラス数	13	12	6	8
クラス木名	証券領域 5	仲介 1	仲介 2	定数表
平均値	4.90	17.30	23.28	16.44
標準偏差	2.20	8.97	10.10	13.70
最小値	4.00	7.38	11.60	6.00
最大値	11.78	36.77	43.77	48.00
クラス数	13	9	11	8

図 3.3に，SystemC の全クラスとクラス継承木ごとに分けたクラスの CNOM に対する CLOC の値を散布図で表した．グラフから，CNOM と CLOC の間に強い正の相関関係を観測できる．相関分析の結果，全クラスにおける相関係数は 0.793，証券領域クラス木で 0.927 の最大値が得られた他，最小の定数表クラス木でも 0.561 が得られた．定数表クラス木の相関係数が小さいのは，CNOM および CLOC の値がともに小さく，値の小さい変動が相関係数に大きな影響が与えているためである．以上から SystemC の CLOC と CNOM の間にも，正の相関があることを統計的に確認できた．

これらのクラス継承木を用いて，帰無仮説 H_0(C値固有) の検定を行う．検定は，有意水準 5%の両側 t 検定で行った．表 3.9に，SystemC の第 14 版の全クラスを対象にして，クラス継承木ごとに求めたクラスの C 値の基本統計量を示す．表 3.10には t 検定の結果から求まった有意水準 P 値を示した．P 値から，観測に用いたクラス継承木は二つのグループに分類できることがわかる．二つのグループとは，証券領域クラス木を構成する 5 つの継承木と，仲介クラス木を構成する 2 つの継承木である．これらのグループ内のクラス継承木の C 値には有意な差を認めることはできないが，グループ間のクラス継承木の C 値には，有意な差が認められる．これは，証券領域クラス木を構成する 5 つの子継承木は，共通の C 値を持っており，また，仲介クラス木に属する 2 つの子継承木も C 値を共有していることを表す．

表 3.10: SystemC のクラス継承木のクラスを対象とした C 値に対する両側 t 検定で求めた有意水準 ; P 値（ただし，証券 n は証券領域 n のクラス継承木を表す）

クラス木	証券 1	証券 2	証券 3	証券 4	証券 5	仲介 1	仲介 2	定数表
証券 1								
証券 2	0.510							
証券 3	0.337	0.483						
証券 4	0.369	0.610	0.752					
証券 5	0.964	0.708	0.389	0.474				
仲介 1	0.003	0.004	0.005	0.004	0.003			
仲介 2	0.000	0.000	0.000	0.000	0.000	0.178		
定数表	0.049	0.053	0.068	0.062	0.050	0.882	0.254	

以上の結果から，帰無仮説 H_0(C値固有) を棄却する．すなわち，クラス継承木の C 値には，クラス継承木間で有意な差異が認められる．

帰無仮説 H_0(C値固有) を検証するにあたり，図 3.3の左下の図でマークをつけたクラスは，仲介クラス木からは除いた．このクラスは 2 つの仲介クラス木の共通のスーパークラスである．スーパークラスの役割については，次節で考察することにする．

2. C 値開発者非依存仮説の検証

SystemC では，二人の開発者が証券領域クラス木に属するクラスを分担して開発した．そこで，人間に依存する差異がクラス継承木の C 値へ及ぼす影響について調査し，C 値開発者非依存仮説の検証を行った．第一の開発者 a は証券領域クラス木 1，2 および 3 を開発し，第二の開発者 b は証券領域クラス木 4 と 5 を開発した．表 3.10に，帰無仮説 H_0(開発者) :「C 値は，開発者間で有意な差異はない」を検証するために行った

両側t検定の結果を示す．検定で用いる有意水準は5%とした．検定結果は，これまでと同様，検定で得られた有意水準のP値で示す．表3.10のP値はいずれも0.05よりも大きいから，帰無仮説 H_0(開発者)を棄却できない．したがって，「C値開発者非依存仮説：C値は，開発者間で有意な差異はない」を検証できた．

3.4 仮説の検証結果と考察

仮説を検証した結果，C値には次の3つの特徴があることがわかった．

- C値はクラス継承木に固有の値である．
- C値はシステムが成長しても安定しており，クラスは，自分が属するクラス継承木のC値を持つように進化する傾向がある．
- 開発者に依存せずに決まる．

ただし，今回行った調査では，同一開発組織の開発者による成果物を対象としているため，これらの成果物は同一のプログラミング規約のもとで開発されたものである．ただし，ここで得られた結論は，プログラミング規約の異なる開発者間でもC値は変わらないことを意味しているわけではない．

クラス継承木の設計はこれまで，形式的に議論されることが多かったが[36]，本研究によって，クラス継承木内のクラスはクラス継承木のC値の制約を受けながら進化する傾向があることを確認できた．この成果から，定量的にもクラス継承木の設計の妥当性を検証できることを明らかにできた．

本研究でC値に用いた行数は単純な改行の回数であるが，行数は同一のプログラミング規約のもとではメッセージ送受信回数との関連が強い．単純に考えると，クラス継承木に分類されたクラスたちが，インスタンス変数の参照と更新だけを行うメソッドだけを持つ受動的なクラスであるとしたら，これらのクラスのC値は一様に小さくなる．また，継承木に分類されたクラスたちが，1回のメッセージ受信をトリガーとして多くのメッセージを送信するクラスであるとしたら，これらのクラスのC値は共通に大きくなると考えられる．したがって，C値はクラスの役割の粒度を表すとみなすことができる．

C値がクラス継承木ごとに固有の値となる原因は，継承の設計指針に依存すると思われる．継承が他のクラスの振る舞いを再利用するために使われる[59]場合，開発者には，継承するライブラリクラスを変更する権限を与えられていないことが多い．そのため，継承を用いた再利用は，新たなクラスの分類方針を設計する作業となる．新たにアプリケーションで定義された継承木では，第2章で示したように，継承関係にあるクラスや兄弟クラスが再設計の対象となる現象を多く観測した．ここで，新たに追加されたクラスのC値がクラス継承木のC値に近付くのは，開発者が継承木に集められたクラスを同程度に複雑な振る舞いを持つようにクラスの役割の粒度を調整しているためと考えられる．

たとえば，メッセージのシグネチャが共通で内部のメソッドがクラスごとに異なる場合，スーパークラスに定義されるメソッドの行数は最小となるが，サブクラスに定義されるメソッドの行数は多くなる．この例では，シグネチャだけがスーパークラスに定義され，メソッドがサブクラスに定義されるからである．各サブクラスに定義されるメソッドはスーパークラスから継承した構造とシグネチャに基づいて実装されるため，それらのメソッドの規模が同程度とならない理由は考えられない．つまり，サブクラスどうしのC値は同程度となる可能性は高い．図3.3の左下の図はこのような設計の例である．この図でマークをつけたスーパークラスのC値は小さく，そのサブクラスである仲介クラス木のクラスのC値はスーパークラスのC値よりも一様に大きく，同程度の値を示している．

また，シグネチャが共通で手続きもほとんど同様の内容である場合，スーパークラスには，共通の手続き部分だけをまとめたメソッドを定義し，サブクラスにはメソッドを再定義してサブクラス固有の手続き部分だけを定義する．この場合，スーパークラスの役割の粒度は大きくなるが，サブクラスの役割の粒度は小さくなるため，やはりサブクラスどうしのC値は同程度となるはずである．

ここで得られた研究の成果によって，クラス継承木のC値を，継承木に新たに追加されるクラスのC値の定量的な設計妥当性確認の指針として応用できることが明らかとなった．今後は，実際にC値を開発支援として使うための評価を進めていきたい．

第 4 章

オブジェクトの進化モデルの構築

4.1 概要

漸進的な開発では，新たに要求される仕様をシステムに取込みながら，クラスやシステムの拡張性を損なわないように注意しなければならない．本研究では，オブジェクトが予測したとおりの進化の過程を経ているか否かを定量的に評価する手段について検討を行った．オブジェクトの進化過程を評価するために，本章ではオブジェクトの種とその進化過程を定量的に違いを示すとともに，オブジェクトの進化過程をソフトウェアアーキテクチャの評価に適用する可能性について議論する．

MVC(Model-View-Controller) アーキテクチャは GUI(Graphical User Interface) を扱うオブジェクト指向システムで一般に導入されているソフトウェアアーキテクチャである．そこで，オブジェクトを種に分類するにあたり，MVC アーキテクチャに基づいたクラスの分類を考えた．本章では，まず第 4.2節でクラスを種に分類し，種と種の環境との関係について議論する．第 4.3節では，アーキテクチャから予想されるオブジェクトの種の進化過程と 3 つのシステム開発事例で実際に観測された進化過程を比較する．続く節ではオブジェクトの種に着目した進化モデルについて議論するとともに，オブジェクトの進化過程をソフトウェアアーキテクチャの評価に適用する可能性についても議論する．

4.2 オブジェクトの種と環境

4.2.1 オブジェクトの種の定義

GUI (Graphical User Interface) を用いるオブジェクト指向システムには，MVC (Model-View-Controller) と呼ばれる階層構造を持ったソフトウェアアーキテクチャを適用することが多い [26]．MVC のうち，View オブジェクトとは，ディスプレイの表示を制御するオブジェクトであり，Controller オブジェクトとは，マウスやキーボード操作を監視するオブジェクトである．また，利用者が Controller を介して操作するオブジェクトは Model オブジェクトと呼ばれる．利用者の操作結果は，View オブジェクトが Model オブジェクトを参照した内容としてディスプレイに表示される．MVC アーキテクチャは，MacAPP，MFC，ET++といった様々なプログラミング言語でも導入され，それぞれの言語で構築されたフレームワークが開発者に提供されている [34]．このアーキテクチャの目的は，全体の要求変更の 40%に及ぶと言われている利用者インタフェースに関わる変更を [35]，View と Controller といったオブジェクトで吸収し，Model オブジェクトに波及させない構造をシステムに持たせることである [18]．

MVC アーキテクチャのような階層構造は，分析工程のモデルにも導入されている．I. Jacobson が提唱する方法論では，オブジェクトを実体オブジェクト，インタフェースオブジェクト，制御オブジェクトの 3 種類に分類する [29]．これらのオブジェクトは，それぞれ Model，View，そして Controller に対応させることができる．

このように，MVC アーキテクチャはオブジェクト指向システム開発において重要なアーキテクチャとなっている．そこで，オブジェクトの種に基づいた進化過程を捉えるにあたり，オブジェクトの分類基準に MVC アーキテクチャを採用することにした．ただし，MVC のうち，View と Controller の関連は強く，いずれも利用者との相互作用が多い点という共通点があるため，これらをまとめてひとつの種と考える．また，汎用クラスの進化についても議論する必要があると考え，本研究では，MVC に基づいた 2 種類のオブジェクトの種にクラスライブラリの種を加えた次の 3 種のオブジェクトについて，その進化過程を議論する．

- 境界領域種：View+Controller クラス群
- 問題領域種：Model クラス群
- 共有領域種：システムで再利用されたクラスライブラリ群

4.2.2 種と環境の関係

3 種類のオブジェクトの種と，オブジェクトの進化に影響を与える 2 つの環境との関係を，進化環境図として図 4.1に示した．図の横軸は利用者環境からの距離を表し，縦軸は技術的環境からの距離を表す．左へ行くほど利用者環境から遠くなり，右へ行くほど利用者環境に近くなる．利用者環境からの距離は，利用者環境変化の影響の受けやすさを概念的に示したもので定量化されてはいない．縦軸は，上へ行くほど技術的環境から遠くなり，下へ行くほど技術的環境に近くなることを表す．技術的環境からの距離は，開発者の設計意図の変化による変更の対象となりやすいことを意味する．したがって，直観的に，境界領域種は利用者環境の変化の影響を受けやすいと考えられるから図中の右下へ，共有領域種は特定の利用者の要求変更の影響を受けることはないので，図中の左上へ位置付けることができる．問題領域種は，MVC アーキテクチャでもアプリケーション依存のオブジェクトとアプリケーション非依存のオブジェクトが混在しているため，個々のオブジェクトの性質によって利用者環境からの距離は異なると予想される．

各環境からの距離は，種に属するオブジェクトの進化の速度にも影響を与える．境界領域種は利用者要求の変更に対応するために早く進化するが，共有領域種は，環境変化の影響を受けないため，ほとんど進化しないか，あるいは緩い進化を起こすだけであろう．問題領域種は，技術的環境の変化，すなわち，開発者が問題領域に対する理解を深めることによって，次第に進化の速度が遅くなると予測される．以上の予測の妥当性を，本論文で取り上げてきた 3 システムのクラスの進化過程によって検証した．

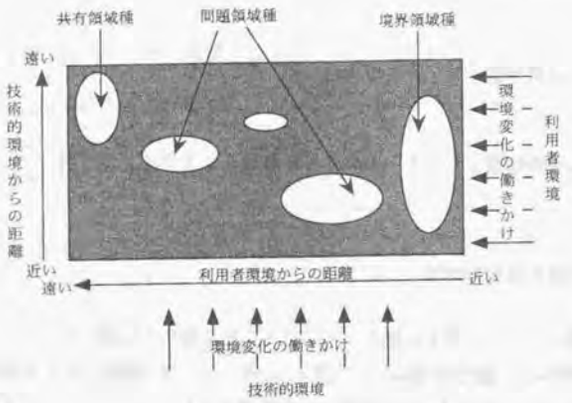


図 4.1: 進化環境図

4.3 種に着目した進化の観測

種別の進化過程を示す前に，3システムの全クラスの進化過程を図4.2に示す．縦軸はクラスの進化率を表し，横軸はシステムの開発期間を表す．ここで，進化率とは，第1章で示した進化率メトリクスを各クラスに適用して求めた値である．図4.2の左上段はSystemA，右上段はSystemB，そして下段はSystemCに属する全クラスの進化過程を表す．図から，SystemA とSystemB のグラフには類似点が多いが，SystemC は他の2システムとは異なる形状をもっていることがわかる．これは，SystemA とSystemB の開発期間中に観測されたクラスの進化率の大きな変動が漸進型開発による利用者環境への適応を表しているのに対して，SystemC のクラスは開発開始から急速に安定状態に到達している．これは，SystemC が落水型開発を採用していたため，開発期間中に利用者環境の変化が発生しなかったためと考えられる．SystemA，SystemB の第4版では進化が緩やかになっているが，これは開発の終盤に至って利用者環境の変化が小さくなったためと考えられる．

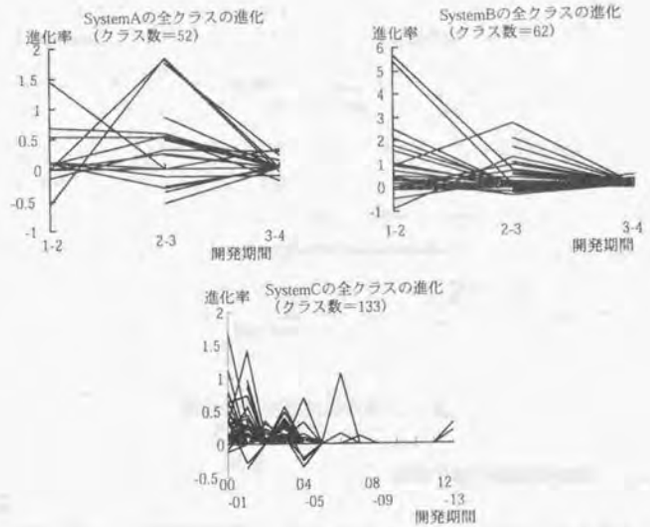


図 4.2: システム全体で観測したオブジェクトの進化過程

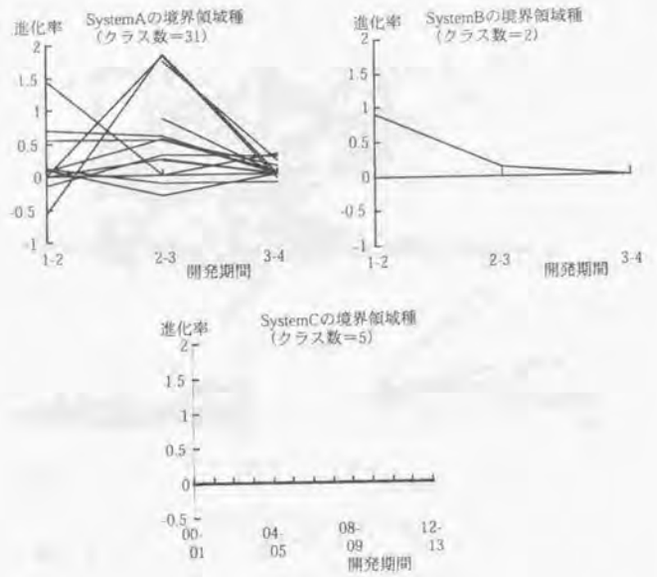


図 4.3: 境界領域種の進化過程

4.3.1 境界領域種の進化過程

3 システムのうち，SystemB と SystemC の境界領域種はコンポーネントを再利用して開発されていた．そのため，SystemB では 2 つのクラスを，また，SystemC では，5 つのクラスを境界領域種に分類できただけである．SystemA では，表示クラス木と編集クラス木に属する最大 31 のクラスを境界領域種に分類できた．

図 4.3の左上段は SystemA，右上段は SystemB，そして下段は SystemC の境界領域種の進化過程を示す．

SystemA の境界領域種には，クラスの進化率に大きな変動を観測できる．また，SystemB では，境界領域種に分類できた 2 クラスのうち 1 クラスに進化

率の変動を観測した．SystemC では，境界領域種の全クラスで進化は観測されなかった．SystemC で観測した現象は，開発期間中に利用者環境の変化が起きていなかったことに起因すると考えられる．

SystemB の第 2 版から第 3 版にかけた開発期間で，全要求変更の 70%が利用者インタフェースの変更要求であったにもかかわらず，SystemA で観測された進化率の変動が観測されていない．境界領域種に変化がないという点では，SystemC と共通している．すなわち，SystemB の境界領域種は，利用者環境の変化の影響を受けていなかったと結論づけることができる．SystemB の利用者インタフェースの開発には，コンポーネントウェアが適用されていたことから，70%の利用者インタフェースの要求変更は，コンポーネントウェアで構築された部分で吸収されたと解釈できる．

コンポーネントウェアをシステム開発に導入することによって，変化が大きい利用者インタフェースは，単純なボタンやスクロールバー，ウィンドウなどのコンポーネントを組み合わせて短期間で構築できるようになった．実際にコンポーネントを組み合わせて構築されたソフトウェアの保守性は高くないが，開発期間を大幅に短縮できるという点で，作り捨て開発に向いている．これは，従来のシステム開発ではシステムの寿命を長くさせることが重要と考えられた時代から，早期開発から作り捨てという利用者環境の変化に合わせた新たなライフサイクルが生まれていることを意味していると考えられる．したがって，SystemB で観測された境界領域種は，コンポーネントウェアを導入することによって，利用者環境の変化をシステム内部に及ぼさないことに成功した例と見ることもできる．

開発者へのインタビューによって，図 4.3に示された SystemB のクラスはコンポーネントを生成するクラスの成長過程を表していることがわかった．しかし，このクラスがコンポーネント用の再利用可能な部品となるまでの進化過程を知るには，さらに長期に渡る観測と他のシステムでの利用状況の観測が必要である．

4.3.2 問題領域種の進化過程

それぞれのシステムで問題領域種に分類できるクラスは以下のクラス木に属するオブジェクトである．

- SystemA
  - 計算クラス木：最大16クラス
  - 物理量クラス木：5クラス
- SystemB
  - 仲介クラス木：最大7クラス
  - 一覧表操作クラス木：最大7クラス
  - 永続化クラス木：最大11クラス
  - 表抽出クラス木：最大7クラス
  - 上記クラス木から独立したクラス：最大13クラス
- SystemC
  - 証券領域クラス木：最大68クラス
  - 仲介クラス木：最大25クラス
  - 定数表クラス木：最大8クラス

図4.4にSystemAの問題領域種の進化過程を示し，図4.5にSystemBの問題領域種の進化過程を示す．それぞれ縦軸はクラスの進化率を表し，横軸はシステムの開発期間を表す．

SystemAの計算クラス木と，SystemBのクラス木から独立したクラスを除くクラスは，いずれも開発が進むにしたがって徐々に進化の速度が緩やかになっている．SystemBの開発期間第2版から第3版にかけて，一覧表操作クラス木で，大きな進化率の変化を示しているクラスは，このクラス木に属するすべてのクラスのスーパークラスである．図4.5では，すべてのサブクラスに共通な進化をスーパークラスが集約することで，サブクラスの進化を緩やかにするという継承の効果を観測できる．

SystemBのクラス木から独立したクラスの進化過程は，SystemAの境界領域種に似た進化過程を示しているが，開発者へのインタビューから，これらのクラスの進化が，偶然的な利用者要求の変更によるものではなく，開発者が，

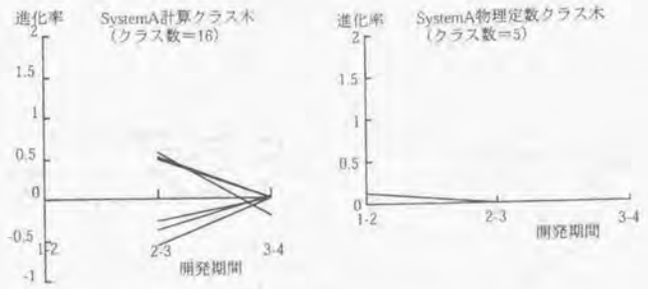


図4.4: SystemAの問題領域種の進化過程

将来のシステム開発に向けて再利用可能な共有領域種として扱えるようにクラスの分割や継承構造の変更を行った技術的要因によるものであることが明らかになった．このグループに属しているクラスの進化過程の詳細は，開発者の設計意図が進化環境図上でどのように働くかという観点から，後で再び触れることにする．

4.3.3 共有領域種の進化過程

SystemBでは，日付け管理を行う2クラスと検索アルゴリズムを保持する11クラスを共有領域種に分類できた．SystemCの共有領域種は，2クラスで，これらのクラスはSystemBで再利用された日付け管理を行うクラスと同じクラスである．SystemAには，共有領域種に分類できるクラスはなかった．

図4.6に共有領域種の進化過程を示す．縦軸は進化率を表し，横軸はシステムの開発期間を表す．図から，いずれのクラスにも開発初期に小さな進化が起き，その後は安定していることがわかる．

この進化のパターンは，SystemCの境界領域種で観測した利用者環境の変化が発生していない場合のオブジェクトの進化パターンと類似している．すなわち，これらのクラスは利用者環境や技術的環境の変化の影響をほとんど受けていないと考えることができる．

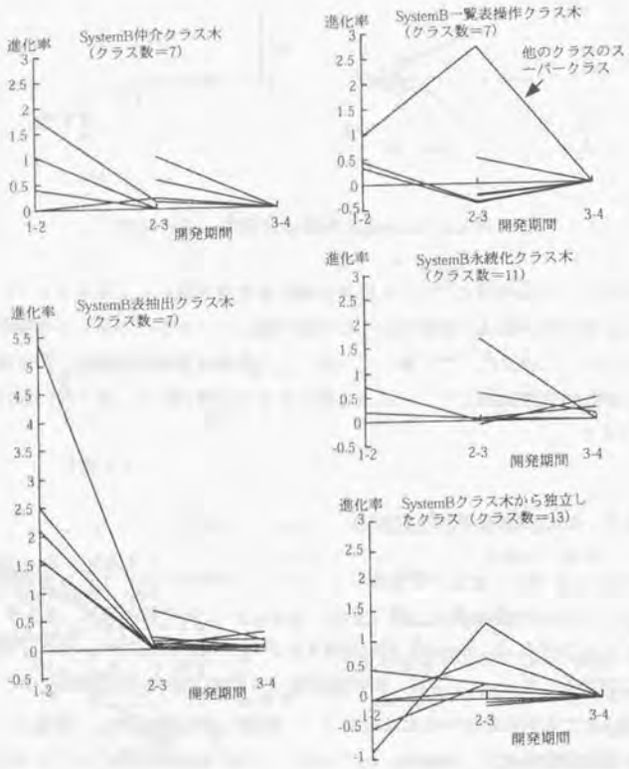


図 4.5: SystemB の問題領域種の進化過程

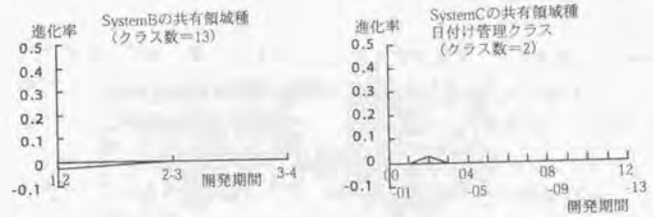


図 4.6: 共有領域種の進化過程

4.3.4 種による進化の違い

以上の観測結果から，多くのオブジェクトは，アーキテクチャから予測された進化パターンを経ることが確認できた．しかし，予測されたとおり，境界領域種には開発期間中に大きな進化率を示すクラスがある一方で，利用者環境の変化が及ばず，その結果，進化しないクラスがあることも確認した．

また，問題領域種は，開発が進むにつれて進化の速度は緩やかとなる傾向があり，この進化の過程で，開発者は問題領域種を安定させるために継承構造の変更や分割を行っていることが明らかとなった．このような進化のパターンは，技術的環境へ適応した結果，オブジェクトが利用者環境から徐々に離れた位置へ移動する過程を表していると理解することができる．そのような進化の過程で，開発者は共有領域種として取り扱いが可能なオブジェクトを発生しようとしていることも確認した．

共有領域種は，新しいシステムに再利用された当初，個々の環境へ適応するために微少の進化が起こるが，その後は安定し，利用者環境や技術的環境の変化の影響を受けることがないことを確認した．

個々の種に属するオブジェクトについて，その進化過程と設計意図とを関連づけ，進化の特徴を詳細にみていくことにする．

SystemA における種の進化

SystemA の計算クラス木に属するオブジェクトは，第1版では定義されていなかった．計算クラス木は，第2版で「熱交換制御のためにいくつかの熱交換アルゴリズムが提供され，シミュレーション設備の種類を増やすように要求された」ため，シミュレーションの複雑化にともなって表示クラスからシミュレーションの計算部分だけを分割して形成されたクラス群である．このような新たな種の出現は，アーキテクチャからは予想できない進化の形態を表す．進化環境図では，右端の境界領域種からより左方に位置する問題領域種が第2版で発生したことになる．

また，第3版から第4版に向かう期間で進化が起きなかったクラスは，境界領域種の場合は46%にすぎなかったのに対し，問題領域種では77%を占めていた．これは，境界領域種が問題領域種よりも利用者環境の変化に敏感に適応していた証拠であろう．以上の結果から，SystemA の計算クラス木に属するオブジェクトを進化環境図に位置付けるとすると，境界領域種よりも利用者環境から遠い左側に位置すると考えられる．これはアーキテクチャから予測できた種の位置づけと一致する．

計算クラス木が開発初期に大きな進化率を示していたのに対して，物理量クラス木に属するオブジェクトは，開発の初期に5クラスのうちの1クラスがメソッド数を1だけ増加させただけで，他の進化を観測できなかった．これは，これらのクラスが物理量という不変の値を保持しているため，利用者環境の変化や技術的環境の変化の影響を受けることがなかったためと考えられる．このようなオブジェクトは，アプリケーション依存のオブジェクトというよりは，物理量計算の問題領域に特化したオブジェクトとみなせる．動的メトリクスを適用した計測結果から，物理量クラス木に属するクラスのオブジェクトは，アプリケーションのクラスからメッセージを受けることはあっても，メッセージをアプリケーションクラスに向けて発信することがないことを確認した．この結果は，これらのクラスが単体で再利用可能なことを意味している．したがって，物理量クラス木に属するオブジェクトは，その進化の性質から進化環境図の左上に位置付けることができ，その仕様から，共有領域種としての取り扱いが可能であると解釈できる．

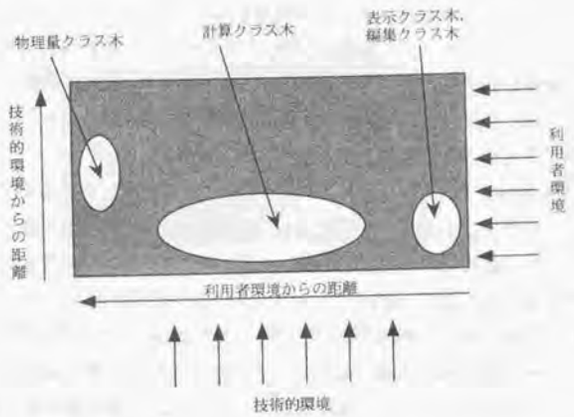


図 4.7: SystemA の種の進化環境図

以上の観測結果から，図4.7に SystemA のオブジェクトの進化環境図を表す．

SystemB における種の進化

SystemB も，MVC アーキテクチャに基づいて開発されたが，SystemA とは異なり，ほとんどの境界領域種のオブジェクトがコンポーネントウェアで開発され，観測対象となった境界領域種はコンポーネントを生成する2クラスであった．2クラスのうち，進化を示したクラスは開発当初に早い進化を示し，継続的に進化しながら徐々に安定していったが，コンポーネントを生成するクラスは，将来，共有領域種に分類できるクラスである．今後，これらのクラスが他のアプリケーション開発でどのように再利用されるかを観察すれば，共有領域種の形成過程を明らかにすることができるだろう．

SystemB の境界領域種はコンポーネントウェアが利用者環境の変化に適応することで，境界領域種の進化は緩やかであったが，次に注目する問題領域種の一覧表操作クラス木では，第2版から第3版にかけて要求された一覧表の操

作に関する要求変更にスーパークラスが適応し，サブクラスの進化を緩やかなままであった．この一覧表操作クラス木に属するオブジェクトは，直接 GUI コンポーネントとメッセージの送受信を行うオブジェクトであるため，利用者環境の変化を受けやすい性質を持つ．これらのオブジェクトを進化環境図へ位置付けるならば，右下方の利用者環境に近い位置にスーパークラスを，より左上方にそのサブクラスを置くことができる．

その他の仲介クラス木，永続化クラス木，表抽出クラス木に属するオブジェクトは，発生した当初に早い進化が現われ，急速にその進化の速度を緩やかにしてゆき，やがて進化が停止する．これらの進化のパターンは，開発者が問題領域の理解を深めることで，MVC アーキテクチャが目指す安定した Model オブジェクトの構築が徐々に完成されていったことを表していると思われる．

次に，以上のクラス木から独立したクラスに注目してその進化過程を見ていこう．これらのオブジェクトの進化過程は，SystemA の境界領域種の進化のパターンと似ている．しかし，これらのオブジェクトの進化は，クラスの分割，統合，継承木からの独立といった，アプリケーション非依存の再利用可能なオブジェクトとなるための進化でり，明らかな技術的環境への適応によるものである．開発者へのインタビューでも，これらのクラスが共有領域種として再設計されたクラスであることを確認することができた．この観測結果から，図 4.5のクラス木から独立したクラスで観測される進化過程は，進化環境図の右下方から左上方へ移動する際に起きた進化の過程と解釈できる．

これらの問題領域種に属するクラスの進化過程と，SystemA で観測された境界領域種から分離して定義された問題領域種の発生状況との共通点から，開発者は，利用者環境の変化を受けて，利用者環境の変化の影響を受けて進化するクラスと進化する必要のないクラスとを切り分け，分割や継承構造の変更などの設計変更を進めながら，より安定したクラスを作り出す設計意図を持っていることがわかる．

では，SystemB の共有領域種についてもその進化過程の特徴をみていこう．SystemB の共有領域種には 2 種類のオブジェクトが含まれている．まず第一の共有領域種は日付け管理に関するクラスで，SystemC でも再利用されたクラスである．第二の共有領域種は検索アルゴリズムを保持するクラスで，システムの開発中第 3 版までは再利用されていたが，第 4 版では削除された．ただ

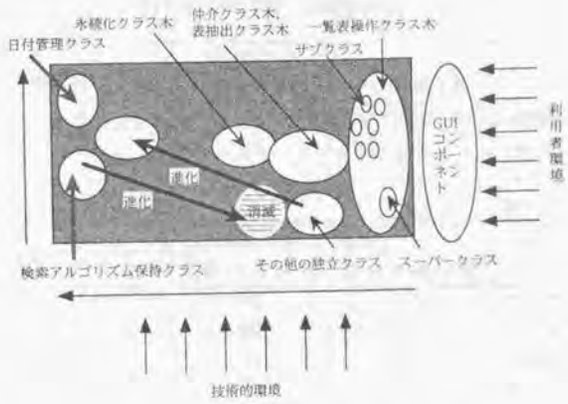


図 4.8: SystemB の種の進化環境図

し，クラスが提供していた一部の役割は他の問題領域種のクラスのメソッドに取り込まれていた．どちらの共有領域種にも再利用を行った初期に，1 メソッド程度の小さな進化が観測されたが，その後の進化は起きていなかった．

第一の共有領域種の進化過程は，共有領域種として予測された進化のパターンを示している．これらのオブジェクトは利用者環境が変化していた期間でも進化していないから，進化環境図の左上に位置付けることができる．しかし，第二の共有領域種の進化過程は，再利用部品を再利用しながらシステムを構築していくボトムアップ型の開発による種の進化過程として解釈できるかもしれない．ここで観測した例は，進化環境図の左方から右方へ移動する際にオブジェクト自身が消滅した例と考えられる．

図 4.8に以上の SystemB の種を進化環境図へ位置付けた図を示す．

SystemC における種の進化

SystemC では，図 4.2に示すように，開発開始後 1 か月でほとんどのオブジェクトの進化が止まっている．

ここで，共有領域種の 2 クラスについて，SystemB で観測された同じクラ

表 4.1: 共有領域種の進化率の分布 (%)

System B: 共有領域種			
	第1版-第2版	第2版-第3版	第3版-第4版
中央値	-0.7	0.0	0.0
最小値	-1.4	0.0	0.0
最大値	0.0	0.0	0.0
クラス数	2	2	3
System C: 共有領域種			
	第00版-第05版	第05版-第09版	第09版-第13版
中央値	0.7	0.0	0.0
最小値	0.0	0.0	0.0
最大値	1.4	0.0	0.0
クラス数	2	2	2

スの進化過程とを比較してみよう．表 4.1に，2つのシステムで観測された2クラスの進化過程を示す．これらのクラスは，同じクラスであるにもかかわらず，一方の SystemB では負の進化率を示し，SysmteC では正の進化率を示している．いずれの変化も微小ではあるが，再利用時に異なった進化が起きたことを表している．クラスを再利用する際，再利用クラスを取り込む側のアーキテクチャと再利用クラスが開発されたアーキテクチャとではアーキテクチャミスマッチと呼ばれる不具合が生じると言われている [17]．これらの微小な進化はアーキテクチャミスマッチを解消するためのものであり，技術的環境への適応と考えられる．

4.4 オブジェクトの進化モデル

ここでは，以上で調査したオブジェクトの種別の進化パターンを整理し，進化モデルを提示することを試みる．以下に観測された各種の進化パターンの特徴をまとめた．

1. 境界領域種の進化パターン

境界領域種に属するオブジェクトの進化は，変化の大きい利用者環境に適応し続けるため，進化率が大きく，継続する．

2. 問題領域種の進化パターン

問題領域種に属するオブジェクトは，利用者環境と技術的環境に適応しながら進化する．また，そのオブジェクトの性質によって，次のような進化パターンに分類することができる．

- 利用者環境の影響を大きく受けるオブジェクト
  境界領域種に似た進化率が大きく継続する進化パターンを持つ．
- アプリケーション固有のオブジェクト
  初期の大きい進化から安定期へ向かう進化パターンを持つ．
- アプリケーション固有のオブジェクトから利用者環境の影響を受けないオブジェクトへ変化するオブジェクト
  進化率の大きい進化後，進化が止まる進化パターンを持つ．進化率の大きい進化は利用者環境の影響を受けないオブジェクトへ変化するための技術的環境への適応による進化である．
- 利用者環境の影響を受けないオブジェクト
  初期の微少な進化があるが，その後に安定する進化パターンを持つ．

3. 共有領域種の進化パターン

初期の微少な進化後，安定期へ至り，進化が止まる進化パターンを持つ．

以上の進化パターンをもとにして，進化環境図を用いたオブジェクトの進化モデルを図 4.9にまとめた．

利用者環境の変化にシステムが適応するためには，進化率が大きくなることを予め予想して開発された部分と進化率が小さくなると予想された部分とを切り分けた構造をソフトウェアのアーキテクチャとして定義する必要がある [4, 53]．MVC アーキテクチャは，階層構造を用いることによって利用者環境の変化を VC のオブジェクトに集中させ，これらのオブジェクトが利用者環境に適応することによって，システムが利用者要求を満足させる構造である．ここ

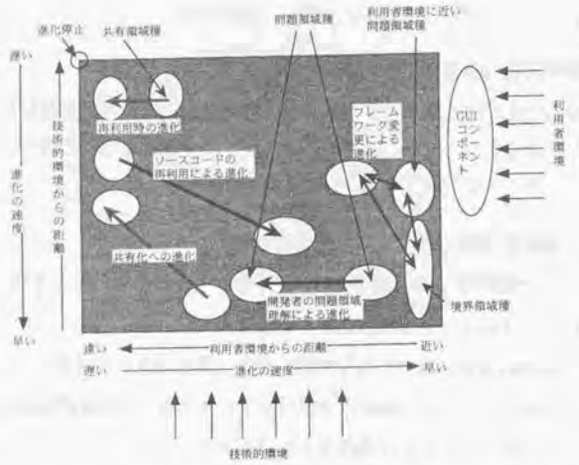


図 4.9: オブジェクトの進化モデル

では境界領域種が進化率の大きい部分となり，問題領域種は進化率の小さい部分となる．もし，MVC アーキテクチャを用いて開発したシステムのオブジェクトが，予想とは異なる進化パターンを持たざるをえないとしたら，それは，開発者の意図的な設計変更の結果であるか，あるいは，そのソフトウェアアーキテクチャがシステムの要求変更の傾向に合致しないことを表していると考えられる．

4.5 考察

本章では，クラスの種による進化過程を調査した結果について議論し，その成果を進化モデルという形でまとめた．さらに多くの事例について調査し，進化パターンと進化環境図中のオブジェクトの位置との関係を検証する必要がある．また，ここで示したオブジェクトの進化モデルを一般化するには，より多くの事例調査が必要であり，ソフトウェアアーキテクチャの妥当性を検証するためにも，他のアーキテクチャを導入して開発された事例を調査する必要がある．

本研究で，共有領域種の，他の種とは異なる進化パターンを定量的に明らかにできたことは，新しい再利用クラスの評価基準を提示できたことになる．これまで，クラスの再利用性に関する議論は，その仕様や再利用に対する開発者の積極性といった人間的側面から行われることが多かったが，共有領域種の進化パターンという側面から，客観的にクラスの再利用性を評価する基準を示すことができた．すなわち，あるクラスが再利用に適する度合いは，そのクラスの進化過程が利用者環境と技術的環境の変化の影響を受けず，進化しないパターンであるかを評価することによって定量的に示すことができる．以上の研究成果から，再利用可能クラスとして登録されているクラスには，その仕様や再利用方法だけでなく，進化の実績データも必要であることが明らかとなった．

第 5 章

オブジェクト進化に基づく組織化過程

5.1 概要

オブジェクトの進化過程を実システム開発事例をもとに調査した結果，ある時点におけるオブジェクトの構造は，オブジェクトの進化の積み重ねによって得られることが明らかとなった．ここでも生物学のアナロジを用いてオブジェクトの組織化について考えてみよう．現在の生物学では，高等な生物が発生するまでには原始生物に始まる何世代にもまたがった進化が必要であると考えられている．ここでオブジェクトの組織化過程を，利用者へシステムが提供されるまでにオブジェクトが形成される過程と定義する．利用者環境に起こる将来のすべての変化を予測することは不可能であるが，利用者環境を時間的変化に伴う変化と利用者の多様化に伴う変化とを区別して考えると，後者の利用者の多様化という変化は分析段階でかなりの程度の予測が可能である．そこで，オブジェクトの組織化過程では，利用者の多様化に適応するための段階的なオブジェクト組織の構築手法を提案する．

従来のOMT法[51]などの分析／設計方法論ではアプリケーションのモデルを構築する際に，問題領域に関する情報を収集し，そこからオブジェクトを抽出する手法が一般的に採用されていた[9]．これらの方法論を実際のシステム分析に適用すると，小規模のシステムでも数十から数百のユースケース[29]から，ひとつのオブジェクトモデルを構築することになる．そのため，モデル化の作業は煩雑となり，適切なオブジェクトを選択する基準を決定する根拠が脆弱であり，分析者の技術力による判断に頼らざるをえないという欠点があった．

また，オブジェクトの組織が様々な利用者要求によって進化する漸進型開発では，時間軸上の変化と利用者の多様化という2種類の利用者環境への適応を考慮する必要があり，頑健なモデルを構築するための技術的環境への適応，すなわち，開発者への負荷が大きくなるという問題があった．しかし，オブジェクトの組織を構築する際に進化過程を模倣した組織化過程を導入し，段階的な組織化を行えば，大規模で複雑なオブジェクトモデルも系統立てたて構築することが可能となり，漸進型開発では，利用者の多様化への対応を軽減することが可能となる．

本研究では，利用者の多様化に対処するために，時間的に変化すると想定される個々の利用者環境に対応して分析の視点を導入することを検討した．この手法では，視点ごとに問題領域を絞り込んでオブジェクトモデル構築した後，順次統合してシステム全体のオブジェクトモデルを構築するため，規模が大きく複雑な問題領域でも，オブジェクトモデルを構築する際に一定の手順に従って進めることが可能となる．

本章では，多視点を用いたモデル化の手法を組織化過程と呼び，その技術的課題について議論し，解決策を提示するとともに，事例を用いて手法を適用する．

構成は次のとおりである．第5.2節で視点を定義し，第5.3節では多視点を用いた分析手法の利点と課題について議論する．課題を解決するための手段については，第5.4節で，オブジェクトの組織構造の特徴を定義した後に検討する．そして，第5.5節で組織化過程の基本方針について説明した後，残りの節で組織化過程を適用した事例を示す．

5.2 モデル化の視点

オブジェクトの組織化過程では，オブジェクトの進化に大きな影響を与える時系列で変化する利用者環境を組織化の視点と定める．組織化の視点とは，要求を発する主体，すなわち利用者がシステムを観察する視点と言うこともできる．本研究では，ある視点に基づいて定義したオブジェクトモデルを利用者モデルと呼ぶ．さらに，これらの利用者モデルを統合してシステムのオブジェクトモデルを構築することから，システムのオブジェクトモデルを統合モデルと

呼ぶ．

5.3 多視点の利点と課題

オブジェクトの組織化過程には2つの特徴がある．第一の特徴はモデル化に視点を導入することであり，第二の特徴は，進化過程を模倣し，利用者モデルを段階的に統合し，利用者の多様性をモデルに取り込むことである．これらの特徴によって，オブジェクトモデルの分析時に，次のような利点を得ることができると考えられる．

- 視点を定めることによってモデル化するオブジェクトの振る舞いや属性の絞り込みが容易となり，モデル化対象の問題領域の境界を明確にできる．
- 視点が複数存在するため，多様な要求変更に対応できる構造をオブジェクトに持たせることが可能となる．
- モデルの統合作業は，オブジェクトが技術的環境へ適応する過程で行われる技術的な設計変更と同様の作業である．オブジェクトの技術的環境への適応が，将来の変更に対する頑健な構造を形成していたように，開発者は統合作業を進めながら要求変更に対して頑健なオブジェクトモデルを検討することができる．
- 視点ごとに分析者を定めれば，それぞれの分析作業を並行して進めることが可能となる．

利用者モデルを構築する際は，モデル化の視点はひとつであるから，従来の分析方法論を適用することが可能である．しかし，利用者モデルを統合する時点で，いくつかの問題が発生する．多視点を用いたモデル化については，データベースのスキーマ定義に関する研究 [64, 3] や，中谷らの研究 [42] があるが，以下の問題について十分な議論が行われてこなかった．

1. オブジェクト間の情報の比較する際に，対応するオブジェクトを個々の利用者モデルから抽出することが困難である．

オブジェクトが異なる視点からモデル化されると，たとえ形式的に同一であったとしても，それらが対応するオブジェクトであるとは限らない．逆に，オブジェクトが形式的に異なる構造を持っていたとしても，それらのオブジェクトが対応しないとは限らない．OMT 法などを用いて定義された利用者モデルから，現実世界の同じ事物や事象をモデル化したオブジェクトを形式的な情報だけから対応づけるのは困難である．したがって，非形式的な現実世界と形式的なモデル世界とを関連づける記述が必要である．

2. オブジェクト間の衝突を発見するために用いる道具がない．

対応づけたオブジェクト間で，視点の違いに由来する仕様の差異を解消するためには，オブジェクトの構造や振る舞いに関する情報を形式的に記述できる道具が必要である．

第一の問題に対処するために本研究では，現実世界とモデル世界とを対応づける記述として，「指示」を導入することを試みる．指示とは，現実世界のオブジェクトとモデル世界のオブジェクトの対応関係に着目し，オブジェクトが指し示す現実世界の事物や事象を限定的に表す認識規則とクラス名を表す述語を関係づけた記述であり [28]，

$$\text{認識規則} \approx \text{指示されるクラス名}$$

で表現される．指示の左辺には，クラスがどのような現実世界の事物や事象を抽象化したかを知るための認識規則を非形式的な自然言語規則を用いて記述し，指示のためのシンボル≈ の後，右辺にクラス名を用いた述語を記述する．

例えば，

$$eng\text{は開発工程を実施する人} \approx \text{技術者}\ (eng)$$

という指示は，技術者というクラスの要素である eng が，開発工程を実施する人を指していることを意味する．特定の問題領域を分析している分析者や問題領域の専門家には，左辺の認識規則を理解して，右辺のクラス名を用いた述語によって指示される対象を識別可能である．もし，指示に定義された認識規則によって現実世界の事物や事象を限定できないとしたら，それは指示の記述が不完全であることになる．

第二の問題に対しては，指示の記述を含めたオブジェクトの性質を定義するためのオブジェクト辞書を用いて対処する．オブジェクト辞書には，利用者モデルに対応する利用者辞書と統合モデルに対応する統合辞書を用意する．統合辞書は，利用者モデルの統合作業と並行して利用者辞書を統合して定義する辞書である．オブジェクト辞書の詳細な構造については，第5.4節で紹介する．

5.4 オブジェクトの組織

5.4.1 オブジェクトの組織構造の特徴

オブジェクトの組織という観点から，現実世界とモデル世界の利用者モデルの対応関係に基づき，利用者モデルが持つ構造の特徴を議論しよう．

組織化過程では，利用者モデルに定義されるクラス群を，利用者オブジェクトの推移的な参照組織の集合として定義する．ここで，参照組織を，注目するクラスが関連を持つクラスの集合と，直系の継承クラスの集合の和集合と定義する．図5.1に，クラス c_x の参照組織を示す．図の表記には OMT 法を用いた．図中で，網かけをしたクラスがクラス c_x の参照組織を構成するクラス群である．

ここで，$R(c_1, c_2)$ はクラス c_1 がクラス c_2 への関連を保持していることを意味する述語，$H(c_1, c_2)$ はクラス c_1 がクラス c_2 の直下のサブクラスであることを意味する述語とする．ただし，R, H には，対称律を仮定しない．H^* を H の推移的閉包としたとき，クラス c_x の参照組織を構成するクラスの集合 $I(c_x)$ は，クラス c_x の関連クラスの集合 $IR(c_x)$ と継承クラスの集合 $IH(c_x)$ を用いて次のように定義される．

$$IR(c_x) = \{c' | R(c_x, c')\}$$
$$IH(c_x) = \{c' | H^*(c_x, c') \vee H^*(c', c_x)\}$$
$$I(c_x) = IR(c_x) \cup IH(c_x)$$

M_u を利用者モデルに定義されたクラスの集合としたとき，M_u の利用者オブジェクトを $c_{vp}(M_u)$ と表す．ここで，M_u の視点はただ一つであるから，M_u には利用者オブジェクト $c_{vp}(M_u)$ がただ一つだけ定義されることになる．

M_u の任意のクラス c は，利用者の視野に含まれる現実世界のオブジェクト

をモデル化したものである．従って，M_uの利用者オブジェクト $c_{up}(M_u)$ とクラス c との間には，直接的，あるいは間接的にメソッドを呼び出す関係が存在すると仮定してよい．このような関係には，クラス間の関連や継承がある．ここでは，これらの関係をまとめて，クラス間のメソッド呼び出し関係と定義する．$RH(c_1, c_2)$ をクラス c_1 とクラス c_2 にメソッド呼び出し関係があることを意味する述語としたとき，

$$RH(c_1, c_2) = R(c_1, c_2) \vee H^*(c_1, c_2) \vee H^*(c_2, c_1)$$

となり，従って，

$$I(c) = \{c' | RH(c, c')\}$$

が成り立ち，RH^* を RH の推移的閉包として，$c_{up}(M_u)$ と M_u の任意の要素 c との間に存在する関係は，

$$\forall c \in M_u \cdot RH^*(c_{up}(M_u), c)$$

とならなければいけない．この式は，利用者モデル内の任意のクラスが，利用者オブジェクトの推移的な参照組織の要素となることを表している．さらにこの式は，利用者オブジェクトから，その参照組織を順に辿って統合作業を進めれば，全クラスの統合作業が完了できることを保証している．

異なる2つの利用者モデルに定義されたクラスが同一であるということは，クラスがモデル化している現実世界の事物や事象が同一であるということである．しかし，視点が異なるとモデル化されたクラスの属性や振る舞い，参照組織といった性質が一致するとは限らないし，粒度が異なる場合もある．そのため，モデルを統合する際には，異なる利用者モデル間に定義された同一のクラスを抽出し，逐一それぞれの性質に起きている仕様の衝突を発見して解消する必要がある．

5.4.2 衝突の種類

利用者モデルを統合する際は，対応するクラスの要素を比較し，図5.2に示した順に衝突を発見して解消する．クラスが持つ性質には依存関係を持っているものがあるため，図5.2には，それぞれの性質の依存関係を矢印で示してある．図中の四角形はクラスの性質を表し，矢印は，各クラスの要素間の制約関

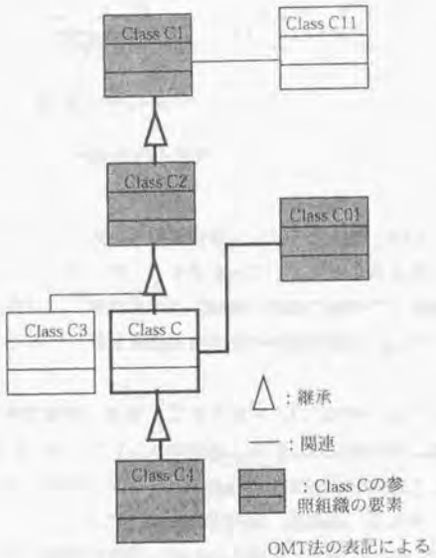


図 5.1: オブジェクト Class C の参照組織

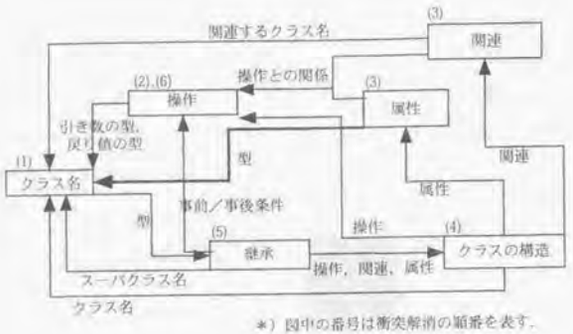


図 5.2: クラスの性質の依存関係

係を表す．たとえば，継承は，クラスの構造，スーパクラス名，操作の事前条件，事後条件が決まることによって決定される [36]．また，図中の番号は，クラスの性質を照合して仕様の衝突を解消する手順を表している．番号が2つつけられているものは，仕様の衝突の検証作業に繰り返しがあることを意味している．

統合時に問題となる衝突には，名前の衝突と構造の衝突が考えられる．名前の衝突とは，異なる対象に対して同じ名前が使われている，あるいは，同じ対象に対して異なる名前が使われているといったクラス間の矛盾をいう．この衝突はクラス名，操作名，属性名，関連名に起こりうる．

構造の衝突は，異なる視点で定義された同じ事物や事象を指示する2つのクラスにおいて，操作や属性といった性質，あるいは参照組織が一致しないことを指す．構造の衝突で，特に継承構造が異なる場合を継承の衝突という．継承の衝突は，オブジェクトの分類方法や抽象度の違いによって発生する可能性がある．

図 5.2 に示した性質の依存関係は，クラス間で発生している性質の衝突を解消する順番でもある．したがって，継承の衝突を解消するのはクラスの性質の衝突解消の最後となる．

以上の衝突を発見する道具として，利用者モデルのクラスの性質を定義する

オブジェクト辞書を用いる．

5.4.3 オブジェクト辞書の構成

オブジェクト辞書には次の2つの役割がある．

- クラスが現実世界の同一の事物や事象をモデル化したものであることを識別するための情報を定義できること．
- クラスの性質間の衝突を発見するための形式的な情報を定義できること．

これらの役割を満足するために，オブジェクト辞書に登録されるクラスには次の項目の情報を定義する．

1. 指示とクラス名

   クラスがデータ抽象した現実世界の事物，事象を指し示す情報で，以下の述語で表現する．

   $$\text{認識規則} \approx \text{指示されるクラス名}$$

2. 操作に関する情報

   操作は，オブジェクト間の契約のもとで実行されるから，辞書には操作のシグネチャと，契約のための表明として事前条件および事後条件 [39] を定義する．これらの情報を比較して，対応する項目の内容が等しければ，それらの操作を同一と判定する．

3. 属性および関連に関する情報

   責任駆動型設計 [62, 63] では，クラスの属性および関連はクラスの責任を果たすために定義されると考える．だから，辞書には属性や関連の意味を説明する情報として，属性の型および関連づけられたクラスの名前とともに，その属性や関連が定義された目的をクラスの操作との関係によって定義する．ここで，属性や関連とクラスの操作との関係が一致すれば同一の属性や関連と判定する．

4. 継承に関する情報

継承を説明するために，辞書には継承するクラスの名前を定義する．継承の衝突を解消する時点で，クラス名の統一が終わっていれば，継承するクラスの名前が一致するだけで同一の継承構造であると判定できる．

利用者モデルを定義する際に使用する利用者辞書に記述する項目を次に例示する．

.....

『オブジェクト名』

- 所有者名: 利用者モデル名
- 指示: 認識規則 ≈ 指示するオブジェクト名
- <参照組織>
  - 継承するオブジェクト名:
    このオブジェクトが責任を果たすために必要な属性と関連として，以下，参照組織を構成する属性数および関連の数だけ繰り返す．
  - 属性名または関連名
    - 属性の型，または参照するクラス名
    - 操作との関係: 属性，または参照関連が定義されている目的
- <操作>

  このオブジェクトが他のオブジェクトに提供することを契約する操作として，以下を操作の数だけ繰り返す．

  操作名（引数の型の並び）：戻り値の型
  - 事前条件:
  - 事後条件:

統合辞書の構造は利用者辞書とほぼ同じであるが，所有者の項目に，統合されたオブジェクトを定義したすべての利用者モデル名を記述する点に違いがある．Rubin と Goldberg らが提唱した OBA(Object Behavior Analysis) では，各作業段階で作成する表に情報の由来を説明する項目がある [50]．複数のツールを用いて分析作業を行う場合には，このような情報の由来を説明する項目が必要である．オブジェクト辞書に記述する所有者項目は，統合モデルのオブジェクトの由来を説明し，利用者モデルのオブジェクトと対応づけるための項目である．統合モデルと利用者モデルを関係付けることができれば，複雑化した統合モデルを理解する際に関連する利用者モデルを参照して理解を助けることが可能であるし，統合モデルのレビューに利用者モデルを使うこともできる．また，統合モデルや利用者モデルを変更した際，所有者項目を手がかりにして，その変更箇所を他のモデルに反映させる作業を進めることができる．

5.5 組織化過程の概要

以上のオブジェクトの組織化過程をまとめると次のようになる．

1. システムを構成するアプリケーションプログラムの利用者を発見し，その利用者が問題領域を観察する視点を分析の視点と定める．
2. 分析者は担当する視点から現実世界を観察し，利用者モデルと利用者辞書を定義する．
3. 利用者辞書を順次選択し，統合辞書に記述されたクラスを指示に基づいて照合し，参照組織の衝突を発見し解消して統合モデルと統合辞書を更新する．
   (3-1) クラス名を統一する．
   (以下の作業をクラスの参照組織を辿りながらクラス数繰り返す)
   (3-2) クラスごとに操作名の衝突を解消する．
   (3-3) クラスごとに属性名および関連名の衝突を解消する．
   (3-4) クラスごとに構造の衝突を解消する．
   (3-5) 継承関係を持つクラスについて，継承の衝突を解消する．

クラス間の衝突を解消する過程は，一方のクラスに新たな仕様を取り込ませる設計作業であるから，オブジェクトが利用者環境の変化に適応する過程で行

われる設計作業と同じである．利用者モデルの統合時には，開発者が問題領域への理解を深めることによって，より適切なモデル構造を発見して定義することも可能である．この作業は，オブジェクトの進化過程では技術的環境への適応で行われる設計作業と同じである．第2で議論したように，オブジェクトは技術的環境への適応の過程で，より利用者環境に適応しやすい構造を取得していく．したがって，段階的な組織化過程を経ることによって，利用者へシステムが公開された後も，より利用者環境へ適応しやすい構造をオブジェクトに持たせることが可能となる．

5.6 事例

本研究で提案した組織化過程の妥当性を検証するために，ソフトウェア開発管理システムの組織化を行った．

5.6.1 システム概要

ソフトウェア開発管理システムの利用者として，プロジェクト管理者，開発者，構成管理者，品質監査者を想定する．各利用者は次の要求を持っている[27]．

- プロジェクト管理者

  プロジェクトの計画立案，要員の割り当て，進捗管理，成果物の検証，妥当性確認のレビューを行い，プロジェクト進捗上の問題発見と問題解決，再発防止のための開発工程の是正を行えるようにプロジェクトに関するデータを進捗にしたがって随時参照したい．

- 開発者

  割り当てられた開発プロジェクトの業務や他の業務をスケジューリングし，開発業務を終了したら速やかに次の作業に取りかかりたい．

- 構成管理者

  プロジェクトの記録や文書，ソフトウェア部品などの成果物を適切に配布し，保管し，廃棄できるように成果物の版管理を行いたい．更に，成

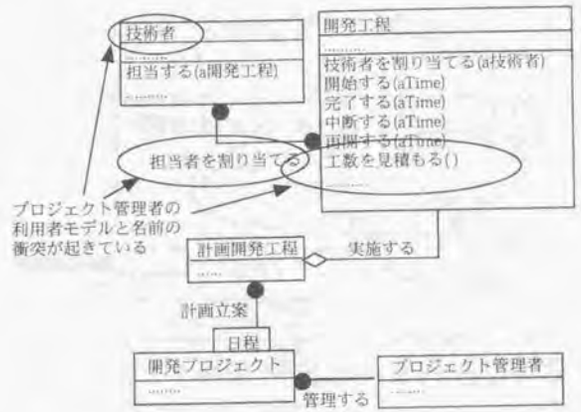


図 5.3: プロジェクト管理者から見た開発工程の参照組織

果物がどのような成果物を参照して生成されたのかといった文書間の関係も管理したい．

- 品質監査者

  構成管理が適切に行われていることを監査しなければならないので，プロジェクトの開発プロセスがその時点の最新の成果物を参照して実施されていたことを調べたい．また，業務の担当者が品質マニュアルに記述されている基準を満たしていることも確認したい．

5.6.2 衝突の発見

クラス名の衝突と解消

組織化手順にしたがって，最初にクラス名の統一を行う．プロジェクト管理者と開発者の利用者辞書を以下に示し，それぞれの利用者モデルに定義された，**開発工程**の参照組織を取り出して図5.3と図5.4に示した．

.....

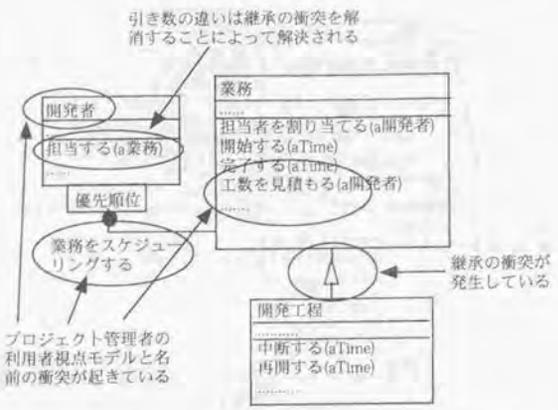


図 5.4: 開発者から見た開発工程の参照組織

『技術者』

- 所有者: プロジェクト管理者モデル
- 指示: *eng*は開発工程を実施する人 ≈ 技術者 (*eng*)

.....

『開発者』

- 所有者: 開発者モデル
- 指示: *dev*はソフトウェアを開発する人 ≈ 開発者 (*dev*)

.....

プロジェクト管理者と開発者の利用者辞書に定義された指示の意味を解釈すると，図 5.3の**技術者**と図 5.4の**開発者**は現実世界の同じ事物を指していることがわかる．そこで，この名前の衝突を解消するために，**技術者**を**開発者**に変更してクラス名を統一する．

1. 属性名および関連名の衝突と解消

プロジェクト管理者のモデルに定義された**開発工程**と**技術者**の間にある関連は，開発者のモデルに定義された**開発者**と**業務**との間の関連と意味が一致するので，関連名に衝突が起きていることになる．ここでは，名前を統一して衝突を解消する．名前の衝突を解消するにあたって，開発者のモデルに定義されていた限定子や多重度を調整する．

2. 構造の衝突と解消

図 5.3と図 5.4に示した開発者モデルの**開発者**およびプロジェクト管理者モデルの**技術者**，それぞれの参照組織を比較すると，属性と操作および関連の集合が一致しない構造の衝突を発見できる．この例では既に関連名の衝突は解消されているから，関連の集合を合成するだけでよい．属性名と操作名についても同様に，それぞれの集合を合成して構造の衝突を解消する．

3. 操作名の衝突と解消

開発工程と**業務**の間の継承の衝突を解消する前に，**開発工程**に発生している操作名の衝突を解消しよう．次に関係する利用者辞書の一部を示す．

.....

『開発工程』

- 所有者: プロジェクト管理者モデル
- 指示:
  proc はソフトウェア開発を管理するために区切られた工程
  ≈ 開発工程 (*proc*)
- <参照組織>
- 継承するオブジェクト名: なし

.....

- <操作>
  工数を見積もる（）: 日数
  - 事前条件:
  - 事後条件:

.....

『業務』

- 所有者: 開発者モデル
- 指示: *task*は，人によって遂行される業務 ≈ 業務 (*task*)
- <参照組織>
- 継承するオブジェクト名: なし

.....

- <操作>

  工数を見積もる（技術レベル）: 日数

  – 事前条件:

  – 事後条件:

プロジェクト管理者はプロジェクトの計画立案を行う際，企業の標準見積り技術を使って開発工程の工数を見積もる．これに対して開発者は，自分の担当業務をスケジューリングするために，自分の技術レベルで必要となる業務の工数を見積もる．したがって，2つのモデルを単純に合成すると，プロジェクト管理者の**開発工程**と開発者の**業務**のサブクラスである**開発工程**は，それぞれ**工数を見積もる**という名前の衝突を起こす操作を持つことになってしまう．この例では，**業務**の**工数を見積もる**を**実質工数を見積もる**に変更し，プロジェクト管理者の**開発工程**に定義されていた**工数を見積もる**を**標準工数を見積もる**に変更する．

4. 継承の衝突と解消

   プロジェクト管理者の**開発工程**と開発者の**開発工程**の属性の集合と操作の集合を合成し，**業務**のサブクラスとして**開発工程**を定義し，開発者とプロジェクト管理者の**開発工程**との間に発生している継承の衝突を解消する．

5. 統合の検証

   図5.5に以上の手順で統合した**開発工程**の参照組織を示す．この統合モデルは開発者とプロジェクト管理者といった利用者モデルを統合したモデルであるから，**開発者**と**プロジェクト管理者**の利用者オブジェクトが含まれている．図5.5では，これらの利用者オブジェクトから統合モデルの検証を始め，参照組織を辿って，他のすべてのクラスの性質が正しく統合モデルに反映されているかを検証する．

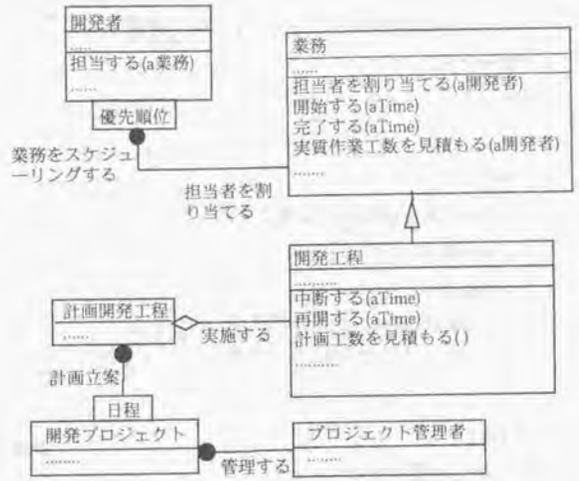


図 5.5: 衝突を解消した結果

6. その他の構造の衝突：関連の衝突と解消

   その他の構造の衝突の例として，構成管理者と品質監査者の利用者辞書の中から，**記録**のサブクラスである**文書**の参照組織を示して説明する．ただし，**記録**とは，版管理されないプロジェクトの成果物を指示するオブジェクトである．

.....

『文書』

- 所有者: 構成管理者モデル

- 指示:
  doc は，版管理されているプロジェクトの成果物 ≈ 文書 (doc)
- <参照組織>
- 継承するオブジェクト名: 記録

……

- <操作>
  参照記録を得る () : 記録の集合
  - 事前条件:
  - 事後条件:
    戻り値 $Rec = \{rec|$ 参照する $(self, rec)\}$
    ($self$は，ここで定義しているオブジェクトを指す)

……

『文書』

- 所有者: 品質監査者モデル
- 指示:
  doc は版管理されている開発工程の成果物 ≈ 文書 (doc)
- <参照組織>
- 継承するオブジェクト名: 記録

……

- <操作>
  参照記録を得る () : 記録の集合
  - 事前条件:
    $\exists proc \cdot$ (新規に生成する $(proc, self)$
    ∨更新する $(proc, self)$)
  - 事後条件: 戻り値 $Rec = \{rec|$
    $\exists proc \cdot$ ((新規に生成する $(proc, self)$
    ∨更新する $(proc, self)) \wedge$ 記録を参照する $(proc, rec))\}$

構成管理者の文書オブジェクトに定義された**参照記録を得る**と，品質監査者の文書オブジェクトに定義された**参照記録を得る**は，事前条件，事後条件は一致しないが，文書と記録は開発工程の成果物を指示しているから，両者の**参照記録を得る**の意味が等しいと解釈できる．したがって，次の関係が成り立つ．

参照する (doc, rec)
$= \exists proc \cdot$ ((新規に生成する $(proc, doc)$
∨更新する $(proc, doc)) \wedge$ 記録を参照する $(proc, rec))$

すなわち，ある文書 doc が記録 rec を参照するということは，記録 rec を参照し，かつ文書 doc を新規に生成するか，あるいは更新する開発工程 $proc$ が存在するということである．したがって，構成管理者の文書間の参照関連を品質監査者の開発工程を経由したモデルで置換すれば，構造の衝突を解消できる．

5.7 考察

以上の手順を経る際に定義したプロジェクト管理者，開発者，構成管理者，品質管理者の利用者モデルを付録Dに掲載した．各利用者モデルを統合して構築した統合モデルを図5.6に示す．

従来の分析方法論を用いると，最初のモデル化のゴールが図5.6のようなモデルを構築することとなる．しかし，組織化過程を用いると，最初のゴールとして限定された問題領域をモデル化対象に設定できるだけでなく，図5.6に向けて，段階的に各利用者モデルを統合する作業を進められるため，モデル化の生産性と精度を従来の方法よりも高めることができると思われる．

技術的には，モデルと現実世界の対応関係を指示によって明示でき，モデルの意味を記述したオブジェクト辞書を用いることによって，視点ごとに定義された利用者モデル間の衝突の発見と解消を容易にできた．また，オブジェクト辞書の所有者項目には，統合モデルと利用者モデルとを対応づける情報を定義するため，統合モデルと利用者モデルの対応関係を管理できる．さらに，レビューに参加する利用者にとって理解可能な利用者モデルが構築されるため，分析の信頼性を向上させることも期待される．

今後は，この組織化過程に準拠した分析手法を開発現場に導入し，漸進的開発に対する効果について研究を進めていきたい．

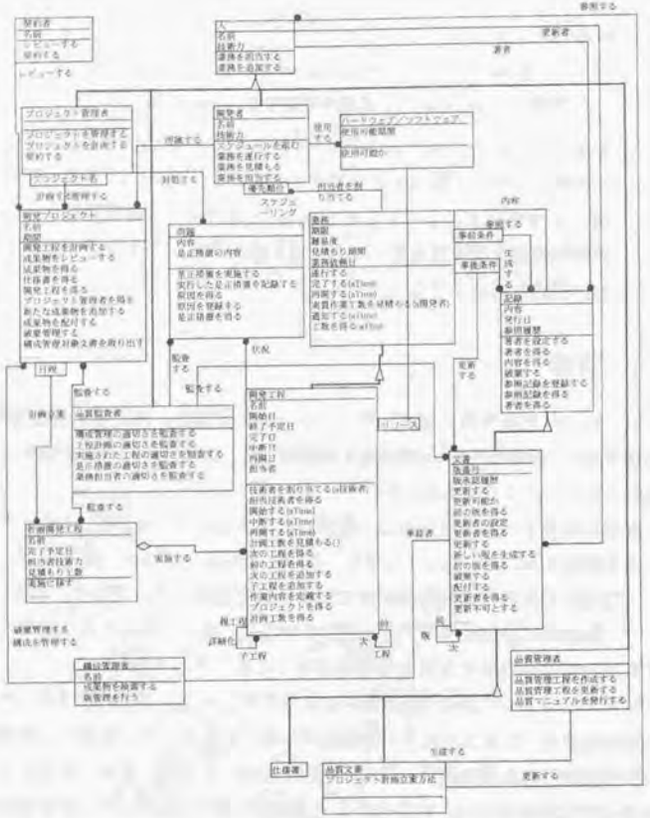


図 5.6: ソフトウェア開発管理システムの統合モデル

第 6 章

まとめと今後の研究課題

6.1 本研究の成果

本研究では，オブジェクトの進化過程を調査し，組織化過程を検討することによって，次のことが明らかとなった．

- クラスの行数，クラスのメソッド数，メソッドの行数の分布は統計的にも安定しており，システムが漸進的に進化しても変わらない．これは，統計モデルを用いて漸進的開発の計画や見積りを行う情報として使えると期待できる．

- その分布の中で，特異な早い進化を示すオブジェクトは，開発者が設計上の問題を持つオブジェクトとして抽出していたオブジェクトと一致した．このことから，オブジェクトの進化過程を観測することによって，設計上の問題箇所を客観的に発見できる．

- クラスは，そのＣ値（１メソッド当たりの行数）がクラス継承木内のクラスのＣ値の平均値に近い値を持つように設計されるか，あるいは設計変更される傾向があり，クラスのＣ値とクラス継承木のＣ値との差異は，クラスの設計上の問題を表わしている可能性が高い．

- クラス継承木のＣ値は開発者によらず，またシステムの進化の影響も受けずに一定であり，各クラス継承木に固有の値として決定される．

- クラスを種に分類したとき，種によって異なる進化過程を観測した．この進化過程はシステムのソフトウェアアーキテクチャから予測できるものと一致する．もし，クラスが，予測した進化過程と異なる進化を起こしている場合，それはシステムのソフトウェアアーキテクチャが利用者環境の変化と対応していないか，あるいは開発者の技術的環境へクラスが適応した過程をあらわしているかのいずれかである．

- オブジェクトの進化過程を組織化過程に適用したところ，従来の手法よりも多面的な情報を持ったオブジェクトモデルを効率的に構築できることが明らかとなった．

6.2 関連研究

M. Lehman と L. Belady は，IBM360 の開発におけるシステムの進化過程を調査し，ソフトウェアの進化法則を定義した [33]．本研究では，Lehman らが指摘したように，クラスやメソッドの中には，利用者の新しい要求を取り入れることで，規模を増大させ続けるものが存在した．しかし，多くのクラスやメソッドは，その規模が増大しない方向に設計変更がなされていた．最近開発されるソフトウェアはサーバ／クライアントの三層モデルのように，階層構造を持たせることでシステムを取り巻く技術進歩の影響を局所化するアーキテクチャが一般的に使われている．これは，ソフトウェアの進化を従来のようにシステムレベルだけで観測するのではなく，システム内のオブジェクトの種に着目して，その進化の特徴を議論しなければならないことを意味していると考える．本研究では，オブジェクトの種という概念を導入し，オブジェクトの進化モデルの構築を試みた．この点で，Lehman らのソフトウェアの進化法則を前進させた議論を展開した．

本研究の調査でも明らかとなったように，オブジェクトは，特定のアプリケーションプログラム内で進化するだけでなく，新たなアプリケーションで再利用される場合にも進化する．ここで，問題となるのは，あるオブジェクトが進化したとき，そのオブジェクトを共有する他のプログラムにも影響が及ぶ場合の対処である．典型的な場合として，データベース上のオブジェクトにマイグレーションを起こさせる必要が発生する場合を指摘でき，この問題に対して

は，現在のオブジェクト指向プログラミングの枠組みでは，オブジェクトの進化を限定したプログラム内だけに収めるという対処ができない．この問題に関して，いくつかの研究が行われている．

W. Harrison らは，上記のオブジェクト指向プログラミングの課題を指摘し，主観的な視点に基づくモデル化の方法としてサブジェクト指向を提起した [19]．W. Harrison らが提案するプログラミング環境では，オブジェクトは識別子と他のオブジェクトとの合成規則によって管理される．合成規則は，オブジェクトの識別子をもとに，一方のオブジェクトに特定のメッセージが送られると，他方のオブジェクトに副作用として予め定義しておいたメッセージを送る役割をもっている．この機構をプログラミング環境に導入することによって，複数のプログラムが一貫性を保ちながら，物理的には別のオブジェクトでも理論的には同一のオブジェクトを取り扱うことができるようになる．

たとえば，ある主観で定義されたオブジェクトが他の主観で定義されたオブジェクトと現実世界の同一のオブジェクトを指示する場合，モデルの外部に定義された合成規則によってモデル間の一貫性が保証される．個々のオブジェクトはシステム内で互いに独立した継承構造，集約構造，振る舞いを持つことができるため，合成規則を書き換えることによって，関連するシステムのオブジェクトを変更することなく，新たなオブジェクトを定義したり変更することが可能である．また，データベース上のオブジェクトを共有しながら，既存のオブジェクトに影響を及ぼすことなく，新たな振る舞いを持つオブジェクトを漸進的に定義することが可能となる．

N. Minsky らが提唱する規則統制アーキテクチャは，各アプリケーションごとに代理オブジェクトを定義し，代理オブジェクトに共通な振る舞いや属性を基本オブジェクトへ定義するアーキテクチャである [40]．このアーキテクチャでは，個々の代理オブジェクトがそれぞれのアプリケーションプログラムの中でメッセージを受け取ると，必要に応じて基本オブジェクトにメッセージを送り，全体としての一貫性が保持されるようになっている．したがって，システムに新たなアプリケーションが追加され，それにともない，既存のオブジェクトに新たな振る舞いが必要となった場合，新たな振る舞いに対応する代理オブジェクトを基本オブジェクトに追加するだけで対応することが可能である．さらに，代理オブジェクトを導入したことによって，オブジェクト指向の継承に

よる可読性の低下を集約構造によって回避できたという利点もある．

プログラミング言語自体の拡張に関する研究も行われている．J. Silling らが提唱したプログラミング言語は，オブジェクト自身に三層構造を与え，代理オブジェクトと基本オブジェクトを一つのオブジェクトの中に実現した [54]．第一階層には，多重化されたインタフェースを定義する．インタフェースの多重化とは，一つのインタフェースに複数のメッセージの組みを定義するという意味である．このオブジェクトと対話を行うオブジェクトは，どのインタフェースを使用するかを予め指定しておく．第二階層は，インタフェースに定義されているメッセージと起動されるメソッドとの関係を管理し，第三階層は，メソッドとインスタンス変数との関係を管理する．この三層構造によって，オブジェクトを進化させるとき，第一階層の関連するインタフェースや，第三階層のメソッドとインスタンス変数の関係を変更するだけで他のインタフェースには影響を与えずにすむ．

いずれの研究も，個々のオブジェクトには多様な進化が許されるが，どのようにオブジェクトの同一性を感知し，一貫性を保つための手段を発見するかについて，議論されていない．本研究で示したオブジェクトの組織化過程は，多視点を用いてオブジェクトの進化過程を分析工程に導入した手法である．この手法ではオブジェクト辞書を用いて，オブジェクトの同一性を発見し，統合時にオブジェクト間に発生していた衝突を解消した．これらの手順は，上記の関連研究で説明した各手法や機構で，オブジェクトの同一性を感知し，一貫性を保つ手続きを定義するためにも有効である．

本研究の組織化過程では，オブジェクトモデルの統合を中心に議論した．オブジェクト指向では，他に機能モデル，動的モデルといった別の技術的視点で構築されるモデルがある．機能モデルはあるシナリオに沿って個々のオブジェクトのメッセージの流れを議論するものであるから，モデルの統合は意味をなさない．しかし，動的モデルについては，オブジェクトの状態遷移図の統合という課題がある．本位田らは，多視点を用いて定義された動的モデルを統合する手法について研究を行った [23]．動的モデルの統合では，主に，新たな状態と事象の発見と，状態の包含関係を発見することに重点がおかれる．分析モデルを構築する際には，本研究の組織化過程を補完する位置付けとなる．

また，本研究では，開発時の手法自体が持つ多視点については議論をしなかったが，B. Nuseibeh らの多視点に関する研究では，動的，静的，機能的な側面を視点ととらえ，視点の異なる仕様書において発生するオブジェクト間の矛盾を発見し解消する方法を提案している [48, 49, 47, 14]．開発プロセス全体を網羅した手法と言われているが，本研究で示した組織化過程が対象とするオブジェクトの抽出，統合といった工程は対象とされていない．

同様の研究として，Nissen らによる M2 モデル（メタメタモデル）の提案がある [46]．M2 モデルとは，多視点による要求をシステムに取り入れるために，仕様書に書かれたモデルのメタモデルから，すべてのモデル化手法に共通な仕様化モデルのメタモデルとして定義されたモデルを指す．M2 モデルでは，システムの仕様書ごとに Agent，Activity，Data，Medium との関係を示すことができるため，各仕様書が記述の対象としている事物や事象の役割分担とシステムにおける立場を概観することが可能となる．したがって，M2 モデルを参照すれば，仕様書に記述された要求の矛盾や抜けを発見でき，そこから新たな要求を発見することも可能となる．

本研究で提案した組織化過程では，利用者モデルを用いて問題領域の専門家とのコミュニケーションを行おうと考えている．H. Nissen らの研究成果は，開発者が技術面からモデルの情報の不備を発見し，そこからさらに新たな要求を発見しようという点で新しい手法であると評価できる．このような手法は，組織化過程における矛盾を発見する手法としても導入する必要があるだろう．

6.3 今後の研究課題

オブジェクトの組織化と進化に関する今後の研究課題を次に挙げる．

- メッセージレベルにおける進化の分析

  メッセージレベルの進化では，メッセージの名前に着目できる [44]．システムの成長にともなう多相メッセージはどのように進化するのか．単語の種類と使用頻度などの解析によって，メッセージレベルの進化を観測することが可能である．

- 再利用ライブラリクラスの進化過程に関する研究

Smalltalk が提供しているクラスライブラリは Smalltak の改版がある都度，その構造やインタフェースに小さな変更がある．このような変更はどのような原因によって発生するものなのか．再利用クラスライブラリの進化過程を明らかにすることで，再利用クラスの開発手法も導くことが可能となるだろう．

- より長いシステム開発事例におけるオブジェクトの進化過程に関する研究

  Tamai らは，ソフトウェアライフサイクルについて，システムの移行の要因を事例から調査した [57]．彼等の研究では，いくつかの企業で，ソフトウェアのライフサイクルを管理し，目標を定めて寿命の予測をたてていることが報告されている．オブジェクトの寿命に関する調査を含めて，より多くの種の進化パターンを収集することは，オブジェクトの進化モデルを精緻化するために重要な研究である．

- 進化パターン抽出のための環境の開発

  本研究では，オブジェクトの進化メトリクスと，オブジェクトの進化過程を定量化するための計測方法および計測結果の解析方法を明らかにできた．今後の計測と分析の生産性を向上させるための環境が必要である．また，進化環境図における技術的環境からの距離に関する概念に関する議論，および観測手段について検討する余地は残されている．

- オブジェクトの組織化過程の充実に向けて

  本研究ではモデル間の矛盾の発見について，事例に基づいた解法を示した．さらに，一般的なモデル間の矛盾の発見と解消手段について検討する余地がある．

参考文献

[1] 青山幹雄: コンポーネント指向ソフトウェア開発方法論，情報処理学会ソフトウェア工学研究会資料, No.111-5, pp.33-40 (1996).

[2] Basili, V. R. and Melo, W. L. : A Validation of Object-Oriented Design Metrics as Quality Indicators, *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, pp. 751–761 (1996).

[3] Batini, C. and Lenzerini, M. : A Methodology for Data Schema Integration in the Entity Relationship Model, *IEEE Transactions on Software Engineering*, Vol.SE-10, No.6, pp. 650–664 (1994).

[4] Boasson, M.: The Artistry of Software Architecture, *IEEE Software*, November, pp. 13–16 (1995).

[5] Boehm, B. W. : *Software Engineering Economics*. Prentice Hall (1981).

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

[7] Booch, G. : *Object-Oriented Analysis and Design with Applications*, second edition, Benjamin Cummings (1994).

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

[9] Coad, P. and Yourdon, E. : *Object-Oriented Analysis*, second edition, Prentice Hall (1991).

[10] Coad, P. : Object-Oriented Patterns, *Communications of ACM*, Vol. 35, No. 9, pp. 152–159 (1992).

[11] Coad, P., North, D. and Mayfield, M. : *Object Models -Strategies, Patterns, & Applications-*, Yourdon Press Computer Series, Prentice Hall (1995).

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

[13] DeMarco, T.: *Controlling Software Projects*, Yourdon Press (1982).

[14] Finkelstein, A. C. W., Gabbay, D., Hunter, A., Kramer, J. and Nuseibeh, B. : Inconsistency Handling in Multiperspective Specifications, *IEEE Transactions on Software Engineering*, Vol. 20, No. 8, pp. 569–578 (1994).

[15] Fowler, M. : *Analysis Patterns -Reusable Object Models-*. Addison-Wesley (1997).

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

[17] Garlan, D., Allen, R. and Ockerbloom, J.: Software Architecture, in Architectural Mismatch: Why Reuse Is So Hard, *IEEE Software*, November, pp. 20–21 (1995).

[18] Goldberg, A.: *Smalltalk-80, The Interactive Programming Environment*, Addison-Wesley (1984).

[19] Harrison, W. and Ossher, H. : Subject-Oriented Programming (A Critique of Pure Objects), *Proc. of Object-Oriented Programming Systems, Languages, and Applications*, ACM, pp. 411–428 (1993).

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

[21] Henderson-Selleres, B. : The Economics of Reusing Library Classes, *Journal of Object-Oriented Programming*, July/August, 1993, pp. 43–50 (1993).

[22] Henderson-Sellers, B. : *Object-Oriented Metrics -Measures of Complexity-*, Prentice Hall (1996).

[23] 本位田真一, 山城明宏 : オブジェクト指向システム開発, 日経 BP 出版センター (1993).

[24] 本位田真一, 青山幹雄, 深澤良彰, 中谷多哉子編著: オブジェクト指向システム分析・設計, 共立出版 (1995).

[25] Humphrey, W. S. : *Managing the Software Process*, Addison-Wesley (1989).

[26] 井上健 : GUI 構築支援システム : GUI クラスライブラリ, 本位田真一, 青山幹雄, 深澤良彰, 中谷多哉子編著: オブジェクト指向システム分析・設計, 共立出版, pp. 84–106 (1995).

[27] ISO-9000-3, *Quality Management and Quality Assurance Standards*, ISO (1991).

[28] Jackson, M. : *Software Requirements and Specifications*, ACM Press (1995).

[29] Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. : *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley (1992).

[30] Jones, C : *Applied Software Measurement*, McGraw-Hill (1991).

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

[32] M. M. Lehman. and L. A. Belady. : *Program Evolution*. Academic Press (1985).

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

[34] Lewis, T. , et al. : *Object-Oriented Application Frameworks*. Prentice Hall (1995).

[35] Lientz, B. P. and Swanson, E. B. : *Software Maintenance Management*. Addison-Wesley (1980).

[36] Liskov, B. and Wing, J. M. : Specifications and Their Use in Defining Subtypes, *Proc. of Object-Oriented Programming Systems, Languages, and Applications*, ACM, pp. 16–28 (1993).

[37] Lorenz, M. and Kidd, J. : *Object-Oriented Software Metrics*, Prentice Hall (1994).

[38] Pree, W. : *Design Patterns for Object-Oriented Software Development*, Addison-Wesley (1996).

[39] Meyer, B. : *Object-Oriented Software Costruction*, Prentice Hall (1988).

[40] Minsky, N. H. and Pal, P. P. : Providing Multiple Views for Objects by Means of Surrogates, *Viewpoints 96: International Workshop on Multiple Perspectives in Software Development (SIGSOFT 96), ACM Press (1997)*. (http://www.cs.city.ac.uk/homes/gespan/vptoc.html).

[41] 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, ソフトウェア工学研究会サマーワークショップ・イン・立山論文誌, 情報処理学会. pp. 145–150 (1995).

[42] Nakatani, T. and Tamai, T. : A Domain Method Integrating Multiple Views, *Proc. of Changsha International CASE Symposium*, China, pp. 205–210 (1995).

[43] Nakatani, T., Tamai, T., Tomoeda, A. and Sakoh, H. : Quantitative Analysis on Evolution Process of Object-Oriented Systems, *Proc. of International Symposium on Future Software Technology '96*, Xi'an, China, pp. 49–56 (1996).

[44] 中谷多哉子, 玉井哲雄, 友枝敦, 酒匂寛. : オブジェクト指向によるシステムの進化を表わすメトリクスの検討ソフトウェアシンポジウム '96 予稿集, SEA, pp. 52–62 (1996).

[45] Nakatani,T., Tamai, T., Tomoeda, A. and Matsuda, H. Towards Constructing an Object Evolution Model *Proc. of Asia-Pacific Software Engineering Conference '97*, IEEE, Hong Kong, pp. 131–138 (1997).

[46] Nissen, H. W., Jeusfeld, M. A., Jarke, M., Zemanek, G. V. and Huber, H.: Managing Multiple Requirements Perspectives with Metamodels, *IEEE Software*, March, 1996, pp. 37–48.

[47] Nuseibeh, B.: Towards a Framework for Managing Inconsistency Between Multiple Views, *Proc. of Viewpoints 96: International Workshop on Multiple Perspectives in Software Development (SIGSOFT 96)*, ACM Press (1997). (http://www.cs.city.ac.uk/homes/gespan/vptoc.html).

[48] Nuseibeh, B., Kramer, J. and Finkelstein, A.: A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification, *IEEE Transactions on Software Engineering*, Vol.20, No.10, pp. 760–773 (1994).

[49] Nuseibeh, B. , Kramer, J. and Finkelstein, A. : Expressing the Relationships Between Multiple Views in Requirements Specification, *Proc. of 15th International Conference on Software Engineering*, pp. 187–196 (1993).

[50] Rubin, K. and Goldberg, A. : Object Behavior Analysis, *Communications of the ACM*, Vol.35, No.9, pp. 48–62 (1992).

[51] Rumbaugh, J., et al. : *Object-Oriented Modeling and Design*, Prentice Hall (1991).

[52] Sharble, R. C. and Cohen, S. S. : The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods, *ACM SIGSOFT Software Engineering Notes*, Vol. 18, No. 2, pp. 60–73 (1993).

[53] Shaw, M. and Garlan, D. : *Software Architecture*, Prentice Hall (1996).

[54] Shilling, J. J. and Sweeney, P. S. : Three Steps to Views: Extending the Object-Oriented Paradigm, *Proc. of Object-Oriented Programming Systems, Languages, and Applications*, ACM, pp. 353–361 (1989).

[55] Shlaer, S. and Mellor, S. : *Object-Oriented Systems Analysis*, Prentice Hall (1988).

[56] Stark, G., Durst, R. C. and Vowell, C. W. : Using Metrics in Management Decision Making, *IEEE Computer*, September, 1994, pp. 42–48.

[57] Tamai,T. and Torimitsu,Y. : Software Lifetime and its Evolution Process over Generations, *Proc. of the Conference on Software Maintenance*, pp. 63–69 (1992).

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

[59] Wegner, P. : Concepts and Paradigms of Object-Oriented Programming, *OOPS Messenger*, Vol. 1, No. 1, ACM Press (1990).

[60] Weller, E. F. : Using Metrics to Manage Software Projects, *IEEE Computer*, September, 1997, pp. 27–33.

[61] Whitty, R. : Object-Oriented Metrics: A Status Report, *Object Expert*, Vol.1, No.1, pp. 35–40 (1996).

[62] Wirfs-Brock, R. and Wilkerson, B. : Object-Oriented Design: A Responsibility-Driven Approach, *Proceedings of Object-Oriented Programming Systems, Languages, and Applications*, ACM, pp. 71–75 (1989).

[63] Wirfs-Brock, R.: Designing Objects and Their Interactions: A Brief Look at Responsibility-Driven Design, Carroll, J. M. ed., *Scenario-Based Design*, John Wiley & Sons (1995).

[64] Yao, S. B., Waddle, V. E. and Hausel, B. C. : View Modeling and Integration Using the Functional Data Model, *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 6, pp. 544–553 (1982).

付録 A

計測値の基本統計量

ここで示す計測値は，各システムにおける

- クラス行数
- クラスのメソッド数
- クラスの変数の数
- 継承の深さ
- メソッド行数

である．それぞれの計測値の基本統計量の変化を示す．ただし，SystemA および SystemB に対しては各版の変化を示すが，SystemC については第 00 版から第 06 版までの約 1 か月半の期間の変化を示す．

表 A.1: SystemA におけるクラス行数 (CLOC) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	97.50	142.85	171.15	166.87
中央値	84	109	100	94
最頻値	120	19	19	14
標準偏差	78.99	134.45	188.68	211.14
最小値	15	15	9	9
最大値	258	474	889	1156
クラス数	16	26	47	52

表 A.2: SystemB におけるクラス行数 (CLOC) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	157.81	230.58	289.85	330.16
中央値	90	111	174	228
最頻値	56	80	271	41
標準偏差	171.01	271.31	327.14	358.50
最小値	9	0	4	23
最大値	745	1519	1902	2090
クラス数	37	81	71	62

表 A.3: SystemC におけるクラス行数 (CLOC) の基本統計量の変化

	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
平均	74.66	92.93	105.70	107.05	115.53	111.67	110.79
中央値	36.5	47.5	49	49.5	55	55	54
最頻値	0	4	4	4	4	4	4
標準偏差	138.68	140.39	156.83	157.80	163.45	159.09	159.63
最小値	0	0	0	0	0	0	0
最大値	820	820	820	820	820	820	839
クラス数	88	118	119	120	120	129	132

表 A.4: SystemA におけるクラスの変数 (NIV) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	2.06	3.92	4.26	3.90
中央値	1	1	2	1
最頻値	0	0	0	0
標準偏差	3.38	6.03	7.00	6.83
最小値	0	0	0	0
最大値	12	24	30	31
クラス数	16	26	47	52

表 A.5: SystemB におけるクラスの変数 (NIV) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	3.14	2.14	2.54	2.73
中央値	1	1	1	1
最頻値	0	0	0	0
標準偏差	4.48	3.88	4.17	4.53
最小値	0	0	0	0
最大値	17	21	21	21
クラス数	37	81	71	62

表 A.6: SystemC におけるクラスの変数 (NIV) の基本統計量の変化

	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
平均	1.55	1.67	1.83	1.82	1.83	1.83	1.80
中央値	0	0	0	0	0	0	0
最頻値	0	0	0	0	0	0	0
標準偏差	2.88	2.93	3.08	3.07	3.09	3.04	3.01
最小値	0	0	0	0	0	0	0
最大値	17	17	17	17	17	17	17
クラス数	88	118	119	120	120	129	132

表 A.7: SystemA におけるクラスのメソッド数 (CNOM) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	10.38	15.65	17.60	16.58
中央値	7	9	11	11
最頻値	0	0	7	7
標準偏差	11.12	16.73	19.32	19.67
最小値	0	0	0	0
最大値	34	63	100	110
クラス数	16	26	47	52

表 A.8: SystemB におけるクラスのメソッド数 (CNOM) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	23.27	30.75	38.10	42.65
中央値	14	18	22	32
最頻値	10	11	7	18
標準偏差	21.87	31.97	36.40	38.60
最小値	4	0	2	3
最大値	94	138	174	181
クラス数	37	81	71	62

表 A.9: SystemC におけるクラスのメソッド数 (CNOM) の基本統計量の変化

	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
平均	7.93	8.98	8.91	10.99	11.72	11.23	11.04
中央値	3	5	5	6	7	6	6
最頻値	3	1	1	1	1	1	1
標準偏差	13.06	12.28	12.26	17.87	18.11	17.71	17.56
最小値	0	0	0	0	0	0	0
最大値	74	74	74	145	145	145	145
クラス数	88	118	119	120	120	129	132

表 A.10: SystemA における継承の深さ (DIT) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	1.69	1.73	1.94	2.02
中央値	2	2	2	2
最頻値	1	2	2	2
標準偏差	0.70	0.67	0.73	0.80
最小値	1	1	1	1
最大値	3	3	3	4
クラス数	16	26	47	53

表 A.11: SystemB における継承の深さ (DIT) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	1.68	1.91	1.83	1.84
中央値	2	2	2	2
最頻値	2	2	2	2
標準偏差	0.58	0.81	0.72	0.73
最小値	1	1	1	1
最大値	3	4	3	3
クラス数	37	81	71	62

表 A.12: SystemC における継承の深さ (DIT) の基本統計量の変化

	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
平均	2.91	3.08	3.06	3.04	3.04	3.01	2.99
中央値	3	3	3	3	3	3	3
最頻値	3	3	3	3	3	3	3
標準偏差	1.38	1.34	1.35	1.36	1.36	1.34	1.34
最小値	1	1	1	1	1	1	1
最大値	7	7	7	7	7	7	7
クラス数	88	118	119	120	120	129	132

表 A.13: SystemA 全体で集計したメソッド行数 (MLOC) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	8.08	8.52	9.09	9.36
標準誤差	0.78	0.77	0.66	0.71
中央値	5	5	5	5
最頻値	4	4	4	4
標準偏差	10.79	16.01	19.53	21.52
最小値	3	3	3	3
最大値	116	261	410	427
メソッド数	193	436	885	927

表 A.14: SystemB 全体で集計したメソッド行数 (MLOC) の基本統計量の変化

	第 1 版	第 2 版	第 3 版	第 4 版
平均	6.78	7.50	7.61	7.74
標準誤差	0.29	0.16	0.15	0.15
中央値	4	4	4	5
標準偏差	8.55	8.22	7.93	7.76
最小値	1	1	1	1
最大値	99	99	99	99
メソッド数	861	2491	2705	2644

表 A.15: SystemC 全体で集計したメソッド行数 (MLOC) の基本統計量の変化

	第 00 版	第 01 版	第 02 版	第 03 版	第 04 版	第 05 版	第 06 版
平均	9.41	10.35	9.72	9.74	9.86	9.94	10.04
中央値	5	5	4	4	5	5	5
最頻値	4	4	4	4	4	4	4
標準偏差	12.90	14.46	13.73	13.76	13.56	13.65	13.71
最小値	2	2	2	2	2	2	2
最大値	168	171	177	177	178	178	178
メソッド数	698	1060	1294	1319	1406	1449	1457

付録 B

観測度数分布に対する幾何分布の導入

次のページから示すグラフおよび表は，各システムにおける

- メソッド行数
- クラス行数
- クラスのメソッド数

について，その度数分布とともに，各計測値の平均値をパラメータとする幾何分布のグラフを重ねたものと，計測値の基本統計量を表わしたものである．計測値をヒストグラムで表わし，幾何分布から求めた理論値を折れ線グラフで示す．ヒストグラムの階級の幅は，メソッド行数は 2 行，クラスのメソッド数は 2 メソッド数，クラス行数は 50 行とした．ただし，ここで観測度数の対象とした標本は SystemA および SystemB は全版，SystemC は最終版とした．これは，SystemA および SystemB についてはクラスとメソッドに変動があり，特別の版を選択する基準がなかったこと，SystemC については，第 00 版から最終版までの経過が連続した開発途中経過のデータであったためである．

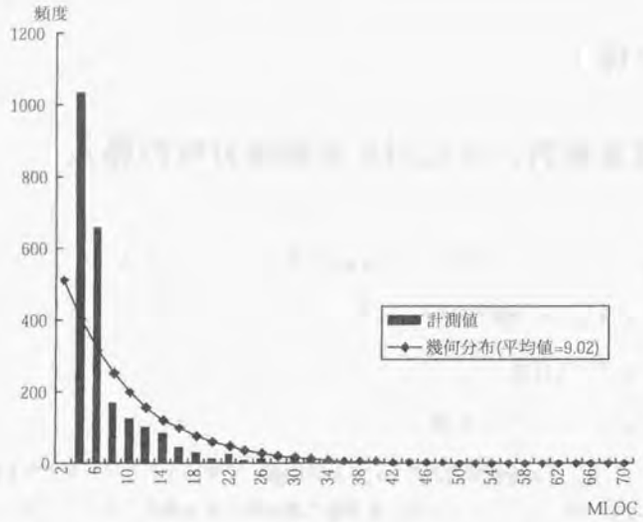


図 B.1: SystemA におけるメソッド行数 (MLOC) の観測度数分布と幾何分布

表 B.1: SystemA におけるメソッド行数 (MLOC) の観測値の基本統計量

平均	標準偏差	最小値	最大値	メソッド数
9.02	19.23	3	427	2435

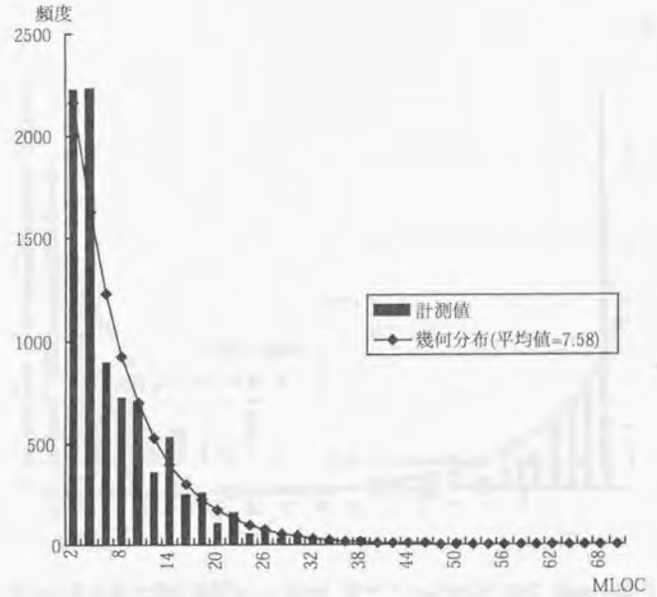


図 B.2: SystemB におけるメソッド行数 (MLOC) の観測度数分布と幾何分布

表 B.2: SystemB におけるメソッド行数 (MLOC) の観測値の基本統計量

平均	標準偏差	最小値	最大値	メソッド数
7.58	8.11	1	99	8778

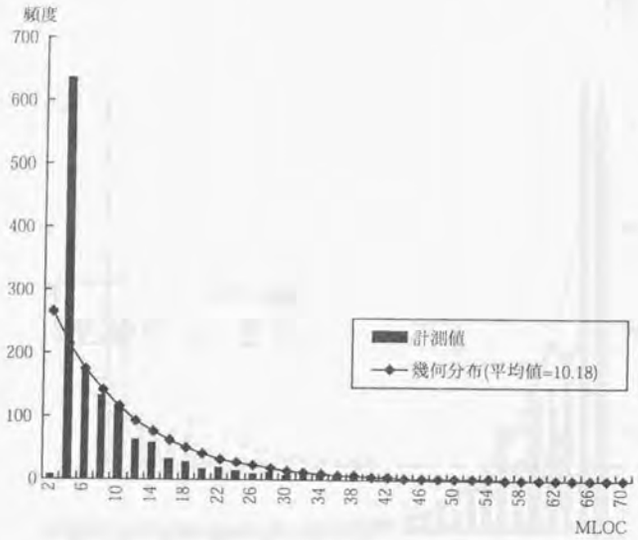


図 B.3: SystemC におけるメソッド行数 (MLOC) の観測度数分布と幾何分布

表 B.3: SystemC におけるメソッド行数の観測値の基本統計量

平均	標準偏差	最小値	最大値	メソッド数
10.18	13.79	2	178	1414

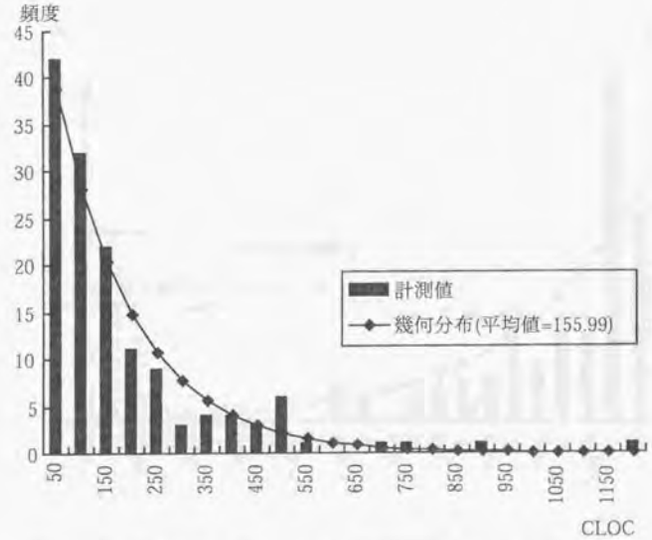


図 B.4: SystemA におけるクラス行数 (CLOC) の観測度数分布と幾何分布

表 B.4: SystemA におけるクラス行数 (CLOC) の観測値の基本統計量

平均	標準偏差	最小値	最大値	クラス数
155.99	179.94	9	1156	141

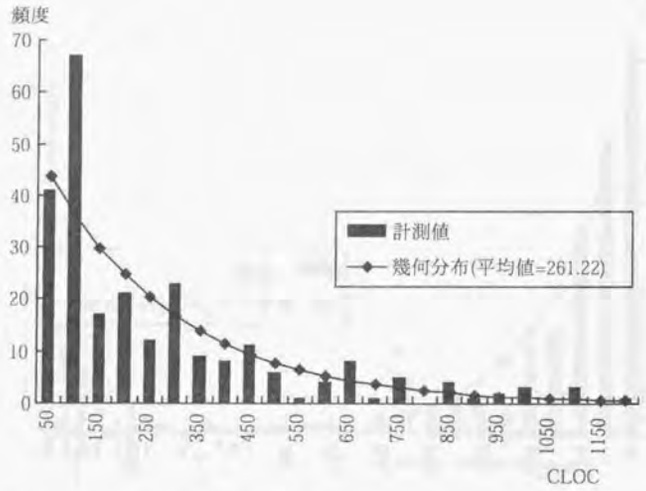


図 B.5: SystemB におけるクラス行数 (CLOC) の観測度数分布と幾何分布

表 B.5: SystemB におけるクラス行数 (CLOC) の観測値の基本統計量

平均	標準偏差	最小値	最大値	クラス数
261.22	303.96	0	2090	251

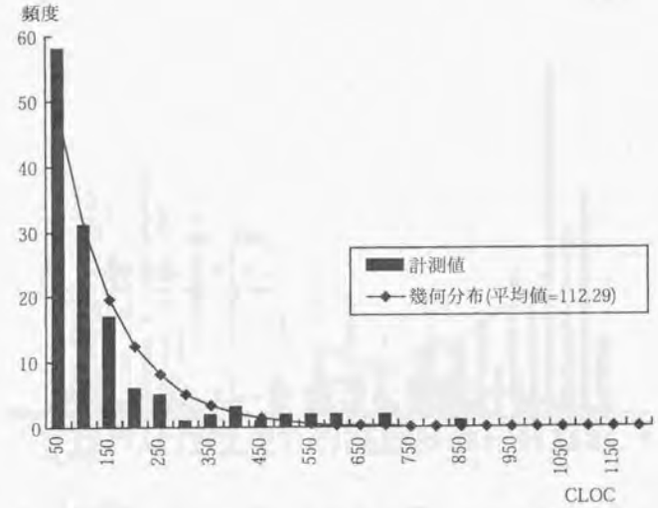


図 B.6: SystemC におけるクラス行数 (CLOC) の観測度数分布と幾何分布

表 B.6: SystemC におけるクラス行数 (CLOC) の観測値の基本統計量

平均	標準偏差	最小値	最大値	クラス数
112.29	159.13	0	839	133

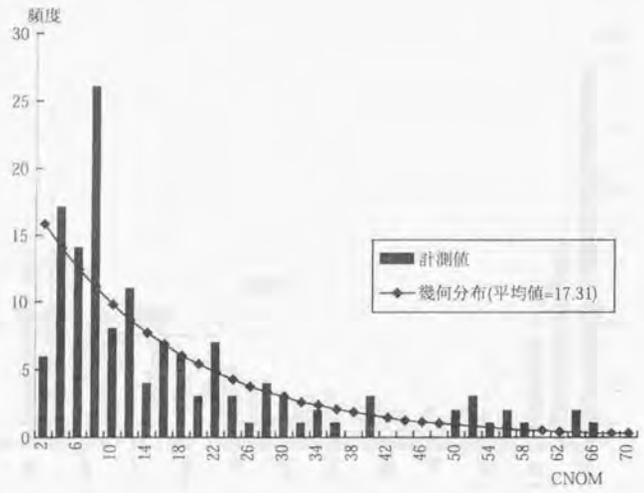


図 B.7: SystemA におけるクラスのメソッド数 (CNOM) の観測度数分布と幾何分布

表 B.7: SystemA におけるクラスのメソッド数 (CNOM) の観測値の基本統計量

平均	標準偏差	最小値	最大値	クラス数
17.31	18.44	2	111	141

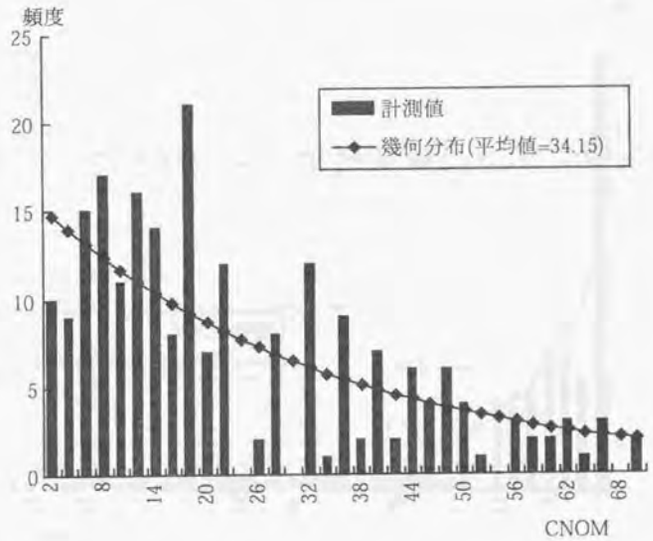


図 B.8: SystemB におけるクラスのメソッド数 (CNOM) の観測度数分布と幾何分布

表 B.8: SystemB におけるクラスのメソッド数 (CNOM) の観測値の基本統計量

平均	標準偏差	最小値	最大値	クラス数
34.15	34.20	0	181	256

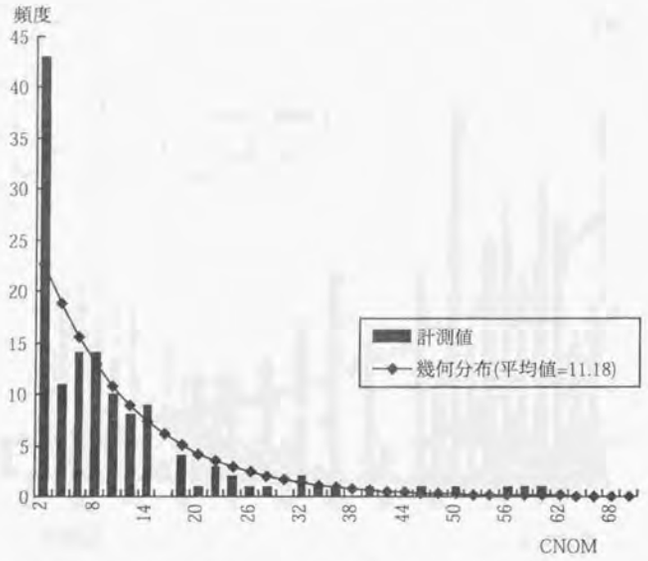


図 B.9: SystemC におけるクラスのメソッド数 (CNOM) の観測度数分布と幾何分布

表 B.9: SystemC におけるクラスのメソッド数 (CNOM) の観測値の基本統計量

平均	標準偏差	最小値	最大値	クラス数
11.18	17.55	0	145	133

付録 C

クラスの性質を表わす計測値の相関分析結果

クラスの性質を表わすメトリクスを用いた計測結果をもとに相関分析を行った．次に 3 システムの各計測値間の相関係数を示す．

表 C.1: SystemA における相関分析結果

	CNOM	CLOC	NIV
CNOM	1.00		
CLOC	0.88	1.00	
NIV	0.86	0.62	1.00

表 C.2: SystemB における相関分析結果

	CNOM	CLOC	NIV
CNOM	1.00		
CLOC	0.95	1.00	
NIV	0.57	0.50	1.00

表 C.3: SystemC における相関分析結果

	CNOM	CLOC	NIV
CNOM	1.00		
CLOC	0.79	1.00	
NIV	0.68	0.51	1.00

付録 D

多視点を用いた組織化事例で定義したオブジェクトモデル

ここで示すオブジェクトモデルは，オブジェクトの組織化の手順を経る際に定義したプロジェクト管理者，開発者，構成管理者，品質管理者の利用者モデルである．

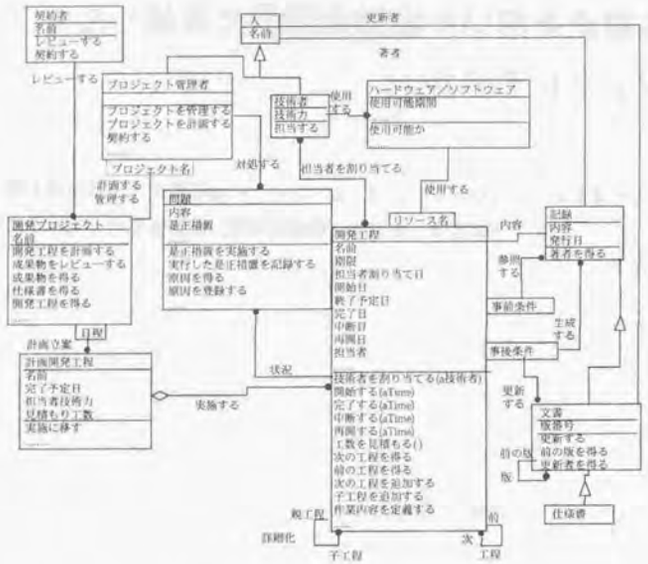


図 D.1: プロジェクト管理者の利用者モデル

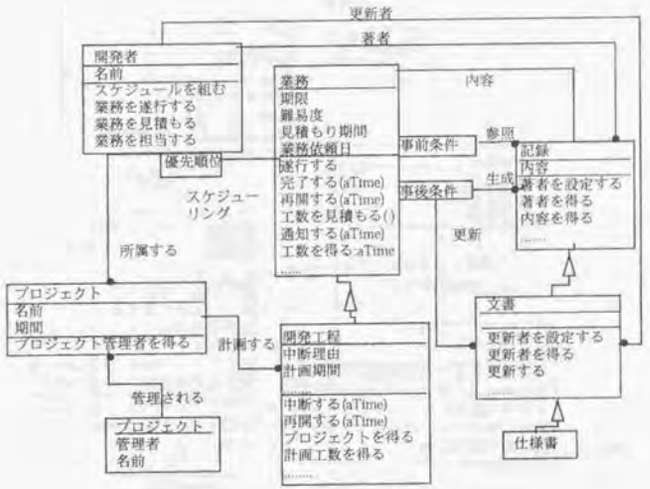


図 D.2: 開発者の利用者モデル

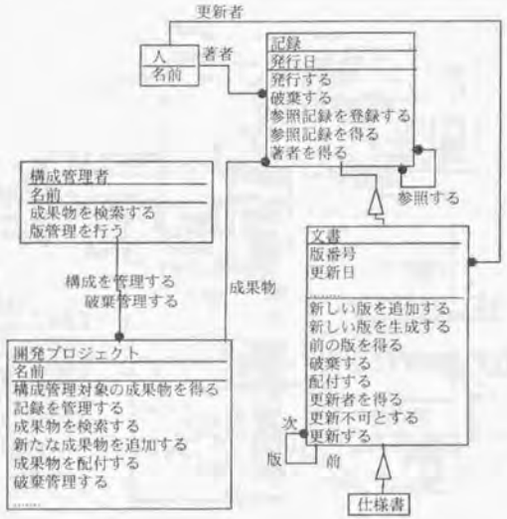


図 D.3: 構成管理者の利用者モデル

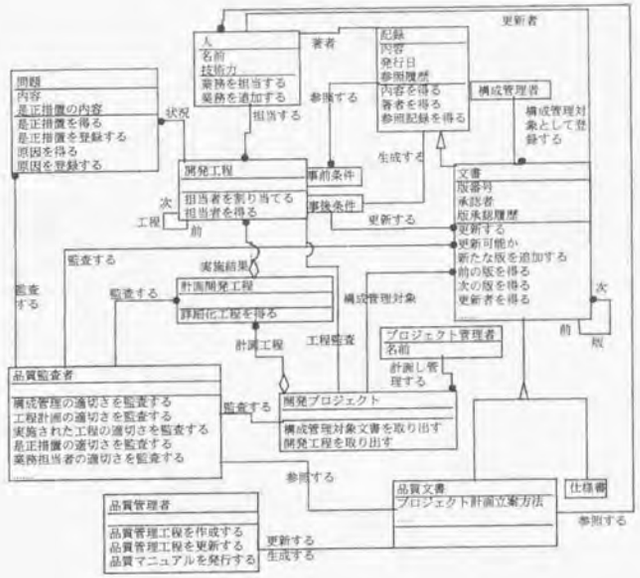


図 D.4: 品質監査者の利用者モデル