静的なオブジェクト指向プログラムに対するインスタンスを考慮した理解コスト計量法

山﨑 直子　掛下 哲郎

大規模なソフトウェアの開発コストを低減するためには理解の容易なソフトウェアを構築する必要がある．そのための第一歩として，我々はソフトウェアの理解容易性を計量するために理解コストを提案している．本論文では静的なOOP (オブジェクト指向プログラム) における理解コスト計量法を提案する．OOPの複雑さを計量するメトリクスとしてはChidamberらが提案したものが有名である．Chidamberメトリクスはクラスのみを計量対象としている．しかし，我々が行った評価実験の結果，静的なOOPの理解容易性はクラスだけでなくインスタンスレベルのプログラム構造に依存することがわかる．そこで，静的なOOPの理解コストを求めるために，その理解過程モデルを提案する．静的なOOPは通常のイベントを処理する際にプログラム構造が変化しないが，OOPの本質的な概念や性質を含む．静的なOOPの理解過程モデルは構造と振舞いの理解過程モデルからなる．構造の理解過程モデルでは，OOPを構成するクラス，インスタンス，複合オブジェクトなどを理解する．これに対して，振舞いの理解過程モデルではOOPのトレースを通じてイベント駆動型の実行，多態性 (polymorphism)，動的束縛を理解する．評価実験で対象とした4種類のOOPに対して，被験者の理解容易性評価値とChidamberメトリクス値が無相関との仮説は有意水準5%では棄却できない．一方，被験者の理解容易性評価値と理解コスト値が無相関との仮説は有意水準5%で棄却される．

1 はじめに

大規模なソフトウェアでは，ソフトウェアライフサイクルコストの約80%が保守作業に費やされている[6]．また，保守作業の約50%が，ソフトウェアの内容を理解してバグの場所や仕様変更に伴う改良場所を特定することに費やされる．したがって，ソフトウェアライフサイクルコストを削減するには，理解の容易なソフトウェアを開発する必要がある．そのためには理解容易性を客観的に計量しなければならない．

理解容易性などのソフトウェアの特性を客観的に計量するためには尺度 (ソフトウェアメトリクス) が必要である．理解容易性には人間によるソフトウェアの理解過程が影響する．しかし，人間の理解特性を考慮したメトリクスは知られていない．

我々は，ソフトウェアの理解容易性を計量するためのメトリクスとして，理解コスト[†1]を提案している[14]．理解容易性を求めるためには，人間によるソフトウェアの理解過程を考慮する必要がある．そのため，理解コストを定義する際にチャンクや体制化と

The understanding cost considering instances for static object oriented programs.
Naoko Yamasaki, 有明工業高等専門学校, Ariake National College of Technology.
Tetsuro Kakeshita, 佐賀大学理工学部, Department of Information Science, Saga University.
コンピュータソフトウェア, Vol.20, No.4(2003), pp.11–24.
[論文]2002年1月31日受付.

†1 紛らわしくない限り，提案したメトリクスの値も理解コストと呼ぶ．

いった認知心理学[5][15]の概念を活用している．

理解コストは理解過程モデルとコスト式から構成される．理解過程モデルはソフトウェアに対する人間の理解過程をモデル化したものである．一方，コスト式は理解過程モデルから理解容易性の度合いを求める計算式である．コスト式は構造化やオブジェクト指向といった設計パラダイムに依存しない．そのため，理解コストの値を求める際，異なるパラダイムのソフトウェアでも同一の計算式で計量できる．

理解過程モデルは設計パラダイムに依存するため，パラダイムごとに定義する必要がある．我々は，これまでに構造化プログラムの理解過程モデルを構築し，このモデルを利用した理解コスト計量法の有効性を評価実験によって示した[14]．現在は，OOPの理解過程モデルを構築している．

プログラマがOOPを理解するには，インスタンスとリンクからなるプログラム構造や個々のイベントによるプログラムの振舞いを理解する必要がある．これらを理解するには個々のインスタンスの状態を理解しなければならない．したがって，我々は，OOPの理解容易性計量にはクラスだけでなくインスタンスの計量も必要と考えている．

既存のオブジェクト指向メトリクスにおいて，インスタンス計量を考慮したメトリクスは知られていない．オブジェクト指向メトリクスの中でも有名なChidamberらが提案したメトリクス[1]も計量対象はクラスのみである．

本論文では，OOPの理解容易性計量にはクラスだけでなくインスタンス計量が必要であることを示す．そして，その考察をもとに静的なOOPに対する理解コスト計量法を提案する．

静的なOOPとはプログラムの実行中にプログラム構造(インスタンス数やリンク構造)が変化しないOOPである．現実的なソフトウェアの中で静的なOOPとみなせるものは数多くある(2.1節参照)．

本論文では，インスタンス計量の必要性を示すために，インスタンス数の異なる静的なOOPに対して被験者がその理解容易性を評価する実験を行う．実験の結果，静的なOOPの理解容易性はインスタンス数に依存することがわかる．また，実験結果とChidamberメトリクス値の間で相関分析を行う．これらの実験や考察を基にして，静的なOOPの理解過程モデルを構築する．静的なOOPの理解過程モデルには，クラス，インスタンス，複合オブジェクト，メッセージ，イベント駆動型の実行，多態性，動的束縛といったOOPの本質的な概念や性質が考慮されている．よって，静的なOOPの理解過程モデルは一般のOOPの理解過程モデルを構築するための基礎にもなる．

静的なOOPの理解過程モデルは構造と振舞いの理解過程モデルから構成される．これらの理解過程モデルを構築する際には，個々のインスタンス状態の理解が必要となる．個々のインスタンス状態の理解を系統的に行うために，インスタンス間でのメッセージ送信をトレースする．

以下，2節では静的なOOPの定義とプログラミングスタイルの仮定を行う．また，静的なOOPに対する理解容易性の評価実験を行う．3節ではChidamberメトリクスの問題点をあげる．4節では理解コストの定義と静的なOOPの理解過程モデルを示す．5節では本計量法がChidamberメトリクスよりも理解容易性を的確に計量できることを示す．6節では本論文のまとめと今後の課題について述べる．

2 静的なOOPの理解容易性評価実験

本節では，OOPの理解容易性を計量する際に，クラスだけでなくインスタンスの計量が必要であることを示す．最初に静的なOOPを定義し，そのプログラミングスタイルを仮定する．その後，実験の概要と実験結果を述べる．

2.1 静的なOOP

静的なOOPを定義するにあたり，OOPの構造と振舞いを決定するイベントについて説明する．イベントは以下の3種類に分けられる．

初期化イベント OOP開始時に1回だけ発生する．このイベントはインスタンスを生成し，インスタンス間のリンクを設定する．

通常イベント 対応するイベントハンドラを起動する．イベントハンドラはOOPを構成するインスタンスに対してメッセージを送信する．この

メッセージ送信によりOOPは処理を行う．

終了イベント OOP終了時に起こる．このイベントはOOP中のすべてのインスタンスを消去する．

通常イベントと終了イベントはオブジェクト指向モデリングに関する教科書[16][12]などで言われているイベントに対応する．初期化イベントは，プログラム起動を仮想的なイベントの実行とみなし，新たに定義している．

静的なOOPは以下のように定義される．

(1) 初期化イベントのみがインスタンス生成およびリンク設定を行える．

(2) 終了イベントのみがインスタンスを消去できる．

よって，静的なOOPでは，通常イベントはOOPの構造を変更できない．しかし，通常イベントはインスタンスの状態を変更できる．

静的なOOPと一般のOOPの違いをまとめる．静的なOOPでは，OOPの実行中にイベントが発生しても，インスタンス生成やリンクの設定が起こらない．よって，静的なOOPの構造は1つしか存在しない．一方，一般のOOPはこれらの制限がないOOPである．一般のOOPでは実行中でもインスタンスの生成やリンクの設定が行える．したがって，一般のOOPの構造は実行中に変化することがある．

一般のOOPが受けるイベントには，処理中にプログラム構造が変更される動的なイベントと，変更されない静的なイベントがある．ここで，保守作業の対象が静的なイベントのみである場合，それ以前に行われた動的なイベントは初期化イベントが行ったものと仮想的に考えることができる．このような条件を満たす場合、静的なOOPとみなせる．例えば，大学などで利用される教務情報システムでは，静的なイベントとして「成績を登録する」,「卒業判定をする」,「教員免許取得判定をする」などがある．一方，動的なイベントとしては「新入生を登録する」,「履修届を登録する」などが挙げられる．ここで，卒業要件が変更された場合の保守作業は静的なイベントのみの修正となる．この保守作業が行われる場合，教務情報システムのプログラムは静的なOOPとみなせる．

実用的なシステムにおいて，保守作業の中心が静的なイベントに対するものであるような場合は多数存在する．例えば，上記の教務情報システムにおいて，「卒業判定をする」処理の修正は頻繁に行われている．これは，卒業要件が文部科学省レベルの規則改定や学科などのカリキュラム改定などに伴って変更されるためである．卒業判定に誤りがあってはならず，その重要度は他のイベントと比べても高い．また，経営情報システム(MIS)やPOSシステムでは，データの取得によってプログラム構造(データ構造)が変更される可能性はあるが，データ解析の際にはプログラム構造は変化しない．よって，データ解析部分を保守する場合，当該プログラムは静的なOOPとみなせる．これらのシステムにおいて，機能的に複雑な部分はデータの取得部分ではなく，統計的技法やデータマイニング技法などを活用して行うデータ解析部分である[13]．

2.2 プログラミングスタイル

プログラミングスタイルはOOPの理解容易性に多大な影響を与える[14]．この影響を除いて理解容易性を考察するために，本論文ではOMT[9]および文献[7]のガイドラインに従い，プログラミングスタイルを以下のように仮定する．

1. クラスは単一の概念に対応する．
2. 変数は単一の情報を格納する．
3. メソッドは単一の機能を実現する．
4. 条件分岐やループの本体は単一機能を実現する．
5. OOPの論理構造はインデンテーションによって表現される．
6. 各メッセージはリンクを通してのみ送信される．

クラスや変数が単一の概念であると判定するには，それらの内容が単一の名詞[†2]で表現できればよい[9]．メソッドなどが単一の機能であると判定するには，機能を単文で記述できればよい．また，仮定3，4を客観的に判定するために，Myers[8]やYourdon[11]らが提案したモジュール強度を利用できる．仮定6は，Demeterの法則[4]に従うことを意味する．

なお，現状では，上記のプログラミングスタイルがすべてのソフトウェア開発現場で守られていることは保証できない．しかし，先進的な企業ではソフトウェ

†2 その名詞に対する修飾詞を含む場合もある．

ア開発の際にコーディング規約を定め，さらにコードの査閲やレビューによって，規約が守られていることを確認している[7]．エクストリームプログラミングにおいてもコーディング規約の重要性が指摘されている[3]．また，文献[4]によれば，任意のプログラムを変換してDemeterの法則を満足させることができる．なお，文献[7]には，現実的なソフトウェア開発を効果的に進めるために，本節で仮定したプログラミングスタイルよりもはるかに厳しいガイドラインが挙げられている．大規模なソフトウェア開発ほどコーディング規約の重要性は増大するので，上記の仮定が守られる可能性は高いと考えられる．

一方，上記のプログラミングスタイルの仮定を守らなかった場合として，1つの変数が異なる目的で多重使用される場合が考えられる．この場合，実行文の中で当該変数がどちらの目的で使用されているかを理解する必要がある．よって，変数の多重使用を含むOOPの理解コストはプログラミングスタイルの仮定を守っているOOPの場合よりも大きくなると考えられる．したがって，本論文で定義した理解コストは，当該プログラムを理解するために最低限必要なコストに対応する．

2.3 実験概要

本実験の目的は，OOPにおいて，インスタンス構造が理解容易性に及ぼす影響を調べることである．実験の被験者は，大学の教職員3名，大学院生7名，学部生10名の合計20名である．すべての被験者はオブジェクト指向に関する知識を持っている．

実験で使用する評価対象プログラムは，以下に示す4種類のプログラムを基本とする．

A: 1次元メッシュを用いた並列ソーティングプログラム．クラスは計算ノードに対応する．

B: 1次元メッシュを用いたN×N行列とN次元ベクトルの乗算を行う並列プログラム．クラスは計算ノードに対応する．

C: LANにおけるルーティングアルゴリズムを用いたデータ転送プログラム．クラスはホスト，ルータ，通信線に対応する．

D: DNSのアルゴリズムを用いた番号とラベルの変換プログラム(分散型)．クラスはホストとサーバに対応する．

上記の基本プログラムに対し，生成されるインスタンス数が異なる主プログラムを用意する．基本プログラムAおよびBに対して主プログラムを5種類，CおよびDに対しては3種類作成した．したがって，実験では全16種類の評価対象プログラムを使用する．

評価対象プログラムはオブジェクト指向言語のソースコードではなく，アルゴリズムで記述されている．これは，実験の被験者が普段利用しているオブジェクト指向言語が異なるためである．また，評価対象プログラムは，2.2節のプログラミングスタイルに従う．ただし，クラス名やメソッド名にはその概念や機能には関連しない名詞を使用する．これは，名前によってクラスなどの内容を推測できないようにするためである．評価対象プログラムの規模は，アルゴリズムのステップ数で100～200程度である．これをJavaやC++などのプログラミング言語で記述するとステップ数は約5倍になる[7]．アルゴリズムを用いることで，より大規模なプログラムを評価対象にできる．

実験は以下の手順で行う．

(1) 各被験者に基本プログラムが異なる3～4種類の評価対象プログラムを割り当てる．

(2) 被験者は各プログラムの理解容易性を10段階で評価する．プログラムの評価は，極めて理解が容易なものを1点とし，理解が困難になるにつれて点数を大きくする．理解不能なプログラムは10点とする．また，被験者が当該プログラムをきちんと理解していることを確認するため，被験者に対して面接を行い，以下の4つの問いに答えさせる．もし，被験者が評価対象プログラムを正しく理解していなければ，そのプログラムの評価値は除外する．

(a) 当該プログラムはどのような機能を持っているか説明せよ．

(b) 各クラスが表している概念を説明せよ．

(c) 各メソッドがどのような機能を果たしているか説明せよ．

(d) インスタンス構成図を示せ．

2.4 実験結果と考察

実験結果を表1に示す．なお，基本プログラムBのインスタンス数15のプログラムの評価値は，担当した被験者が全員理解不能(評価値10)であったため，評価から除外する．

表1の各基本プログラムにおいて，異なるインスタンス数のプログラムに対する評価値の平均が一致する確率(p値)を，等分散を仮定するt検定を用いて計算する．基本プログラムが同一でインスタンス数が隣り合う評価値集合間のp値は11通り存在する．これらのp値は0.26と1の間に分布しており，そのうち5通りは0.5を下回る．また，3.2節で述べるように，インスタンス数に依存しないChidamberらのメトリクスの各値と評価値平均が無相関との仮説は統計的に棄却されない．これらの事実に基づいて，各評価対象プログラムに対する被験者の評価値はインスタンス数に依存することが示される．

図1〜3に示す評価対象プログラムのインスタンス図も参考にして，OOPの理解容易性評価に対する考察を行う．

表1 被験者の評価値平均

基本プログラム名	インスタンス数	評価値平均
A 並列ソーティング	3	4.25
	6	3.25
	9	2.40
	16	3.00
	24	3.00
B 並列行列計算	3	6.00
	5	4.33
	7	5.00
	11	9.00
C ルーティング	8	4.80
	15	6.00
	26	6.00
D DNS	8	4.50
	16	4.80
	28	5.25

図1 評価対象プログラムA,Bのインスタンス図
(A,B共通，インスタンス数3)

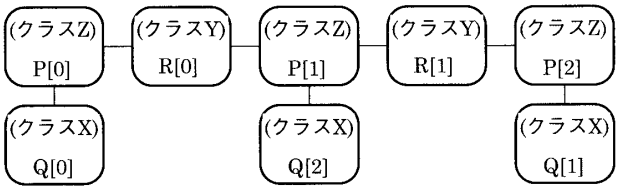

図2 評価対象プログラムCのインスタンス図
(インスタンス数8)

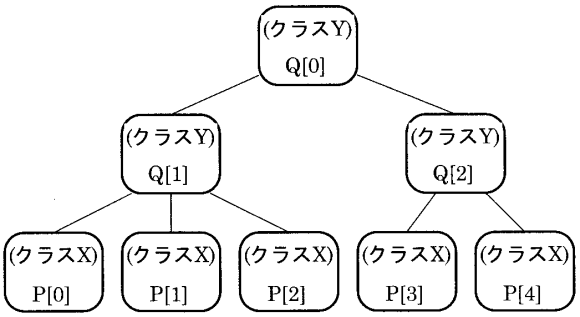

図3 評価対象プログラムDのインスタンス図
(インスタンス数8)

図1はインスタンス数3の評価対象プログラムA(並列ソーティング)とB(並列行列計算)のインスタンス図である．両者はインスタンス数もリンク構造も同一である．しかし，表1の評価値平均より，被験者は並列行列計算の方が並列ソーティングよりも理解が難しいと評価している．被験者に対する面接の結果，並列行列計算の方がイベント(メッセージ送信)によるOOPの振舞いの理解が難しいことがわかった．よって，理解容易性を計量するメトリクスには，OOPの振舞いの理解過程を考慮する必要がある．

また，表1によると，基本プログラムA，Cにおいて，インスタンス数が多い場合には理解容易性の評価が同じである．被験者に対する面接より，プログラム構造および振舞いに存在する規則性が理解を容易にすることがわかった．よって，理解容易性を計量するメトリクスには，OOPの構造および振舞いに存在

する規則性を考慮する必要がある．

また，表1より，基本プログラムA，Bに対しては，インスタンス数が少なくても理解が難しいと被験者は判断している．この原因について考察する．

被験者に対する面接により，被験者がOOPを理解する場合には，個々のインスタンスの処理を理解し，その後にプログラム全体の機能を推測していることがわかった．基本プログラムA，Bはいずれも並列アルゴリズムである．被験者20名中17名は並列アルゴリズムの知識を持たないため，個々のインスタンスからプログラムの機能を理解することは難しい．このような場合には，被験者は入力データを各自与えてプログラムをトレースし，入出力データを関連づけることでプログラムの機能を推測する．しかし，インスタンス数が少ない並列プログラムでは入出力データ数も少ないため，機能の推測が難しくなることがある(例：インスタンス数3の基本プログラムAは入力された2つの整数値を入れ換えるだけに見える)．

以上の考察から，機能が不明なOOPでは，インスタンス数をある程度増やすことで入出力データ間の関連を推測しやすくなると考えられる．一方，インスタンス数を増やし過ぎるとプログラム内の相互作用が複雑になるため，理解が難しくなる．我々は，両者の間に理解しやすいインスタンス数の領域があると予想している(例：基本プログラムBのインスタンス数5の場合)．

3 Chidamberメトリクスの問題点

ChidamberメトリクスはOOPの複雑度計量のためのメトリクスとして頻繁に利用されている．本節では，Chidamberメトリクスで理解容易性の計量が可能かどうか前節の実験を基に検証し，その問題点を挙げる．

3.1 Chidamberメトリクス

Chidamberらは6種類のメトリクスを提案している[1]．これらのメトリクスはクラスを計量対象とし，インスタンスの計量は行わない．以下に，6種類のメトリクスを紹介する．

WMC (Weighted Method per Class)：n 個のメソッドを持つクラス C がある．各メソッドは $M_1, \cdots, M_n$ で表す．各メソッドの複雑度を $c_1, \cdots, c_n$ で表す．この時，WMC $= \sum_{i=1}^{n} c_i$ である．各メソッドの複雑度が等しければ，WMC $= n$ となる．

DIT (Depth of Inheritance Tree of a class)：継承木の根から対象クラスに至るまでの経路の長さである．多重継承している場合は，より長い方を対象とする．継承していないクラスのDITは0とする．

NOC (Number of children)：クラスから直接派生しているサブクラスの数である．

CBO (Coupling Between Object classes)：対象クラスと結合しているクラスの数である．

RFC (Response For a Class)：クラスのメソッド数とそのメソッドによって呼び出される他クラスのメソッド数の和である．

LCOM (Lack of Cohesion in Methods)：同一クラスのメソッドにおいて，2つずつのすべての組合せを考える．それらの組合せにおいて，各メソッドが参照するインスタンス変数の集合に共通するものがない組の数から，共通するものがある組の数を引いたものをLCOMの値とする．ただし，計算結果が負の値であれば，LCOMは0とする．

3.2 Chidamberメトリクスを用いた理解容易性評価

本節では，Chidamberメトリクスで理解容易性が評価できるか検討する．そのために2節の実験で使用した評価対象プログラムのChidamberメトリクス値を計算し，被験者の評価値平均とChidamberメトリクス値の相関分析を行う．また，その結果について考察する．

表2に評価対象プログラムに対するChidamberの各メトリクス値，およびメトリクス値と被験者の理解容易性の評価値平均との相関係数を示す．ただし，評価対象プログラムはサブクラスを含まないので，DITおよびNOCメトリクスは用いない．

表2において，Chidamberメトリクスは基本プロ

表 2 被験者の評価値平均と Chidamber メトリクス値との相関係数

基本プログラム名	インスタンス数	評価値平均	WMC	CBO	RFC	LCOM
A 並列ソーティング	3	4.25	24	0	4	0
	6	3.25				
	9	2.40				
	16	3.00				
	24	3.00				
B 並列行列計算	3	6.00	26	0	8	16
	5	4.33				
	7	5.00				
	11	9.00				
C ルーティング	8	4.80	50	4	19	4
	15	6.00				
	26	6.00				
D DNS	8	4.50	36	2	14	0
	16	4.80				
	28	5.25				
評価値平均と各メトリクスの相関係数			0.33	0.00	0.20	0.49

グラムのインスタンス数が異なっていても，各メトリクス値は同一である†3．これは Chidamber メトリクスがクラスに対する計量のみを行っているためである．しかし，2 節より，被験者の理解容易性の評価値平均はインスタンス数に依存する．よって，Chidamber メトリクスはインスタンス数による理解容易性の違いを説明できない．また，表 2 において，各メトリクス値と被験者の理解容易性の評価値平均との相関はどれも低い．統計的検定を行った結果，評価値平均と Chidamber の各メトリクス値が無相関であるとの仮説は有意水準 5%ではどれも棄却されなかった．

以上より，Chidamber メトリクスは理解容易性計量には適さない．同様に，オブジェクト指向特有の概念に対応した既存のメトリクス [2] の多くがクラスのみを計量対象としているため，理解容易性計量には適さない．

†3 表 2 では WMC 値をメソッド数として計算している．WMC において各メソッドの複雑さを求める方法は他にもある (例えば，参考文献 [10] など) が，それらの方法を用いたとしても結論は変わらない．

4 静的な OOP の理解コスト計量法

2 節により，理解容易性計量にはクラスだけでなくインスタンスを考慮する必要がある．また，OOP の構造および振舞いに規則性がある場合，理解が容易になることも考慮する必要がある．そこで，これらを考慮した理解容易性計量のためのメトリクスを提案する必要がある．

本節では静的な OOP の理解コスト計量法を提案する．最初に理解コスト計量法の定義について説明する．次に，静的な OOP に対する理解過程をモデル化する．また，構築した理解過程モデルに規則性がある場合の理解コスト計算についても述べる．

4.1 理解コストの定義

理解コストはソフトウェアの理解容易性を計量する

ためのメトリクスである．理解コストは理解過程モデルとコスト式から構成される[14]．

理解過程モデルは，技術者によるソフトウェアの理解過程をモデル化したものである．理解過程モデルは木構造で表される．木構造の節点はチャンクに対応する．チャンクとは何らかの意味で単一のまとまりを示す情報のことで，トークン，文，関数，クラス，インスタンスなどのソフトウェアのさまざまな構成要素に対応する．木構造の枝は体制化に対応する．体制化とは下位レベルの節点 (構成チャンク) の集合を 1 つ上のレベルの節点 (高水準チャンク) としてまとめて理解することである．体制化の過程は人間の理解過程に対応する．

コスト式は理解過程モデルから理解コストの値を求める計算式である．n 個の構成チャンク $u_1, \cdots, u_n$ からなる集合を体制化して得られた高水準チャンク U の理解コスト $C(U)$ は以下のように定義される．

$$C(U) = C_R(n) + \sum_{i=1}^{n} C(u_i) + 1 \quad (1)$$

$$C_R(n) = \begin{cases} n & (n \leq 7) \\ C_R(n-1) + n - 6 & (n > 7) \end{cases} \quad (2)$$

式 (1) の右辺において，第 1 項の $C_R(n)$ は集合内のチャンク間の関連に対する理解コストである．第 2 項は各構成チャンク u_i の理解コストの合計である．u_i が基本チャンクであれば $C(u_i) = 0$ である．各 u_i の体制化が完了していれば $C(u_i)$ は定数となる．なお，同一チャンクに対する体制化は重複して行わないものとする．

上記のように，コスト式は 1 回の体制化に要するコストを求める式として定義されている．したがって，ソフトウェア全体の理解コストの値は，理解過程モデル中の体制化ごとにコスト式を繰り返し適用することで求められる．

コスト式は設計パラダイムに依存しない．そのため，理解コストの値を求める際，異なるパラダイムのソフトウェアでも同一の計算式で計量できる．一方，理解過程モデルは設計パラダイムごとに定義される．そのため，各種パラダイムごとに理解過程モデルを構築する必要がある．

4.2 静的な OOP の理解過程モデル

OOP を理解するためには，個々のイベントによる OOP の振舞いと，それによって変化する OOP の構造を理解する必要がある．静的な OOP の場合，取り得る構造は 1 つだけである．この構造は，初期化イベントが起こった直後のプログラムの状態 (初期状態) を理解することによって理解できる．

静的な OOP の理解過程モデルを図 4 に示す．図 4 の木構造では，各節点に対応するチャンクがその親節点に対応する高水準チャンクの構成要素になる．このため，前節の定義にしたがって，各高水準チャンクの理解コストが計算できる．なお，以下に示す木構造中の節点について，四角形はソースコードの構成要素，丸四角はソースコードの構成要素以外で理解過程に必要なチャンクを示す．また，着色された節点は基本チャンクを表す．

OOP の構造や振舞いを理解するためには，ソースコードを理解しておく必要がある．以下では，最初にソースコードの理解に必要なメソッドおよびクラスの理解過程をモデル化し，その後で OOP の構造と振舞いの理解過程をモデル化する．

メソッド

基本チャンクは実行文や変数宣言を構成するトークンとする．4.1 節より，基本チャンクの理解コストは 0 である．これにより，n 個のトークンから構成される文の理解コストは $C_R(n)$ となる．また，アルゴリズムステップは，これを構成する文の体制化により 1 つの高水準チャンクとして理解される．入れ子の文は，入れ子の内側から外側へ体制化を繰り返し行うことによって理解される．入れ子のアルゴリズムステッ

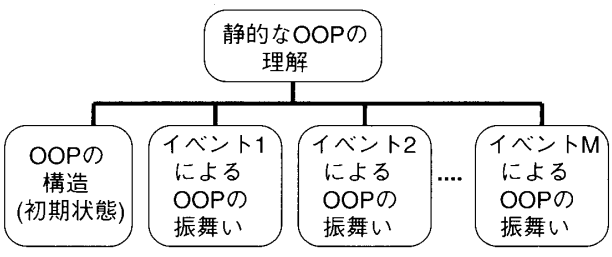

図 4 静的な OOP の理解過程モデル

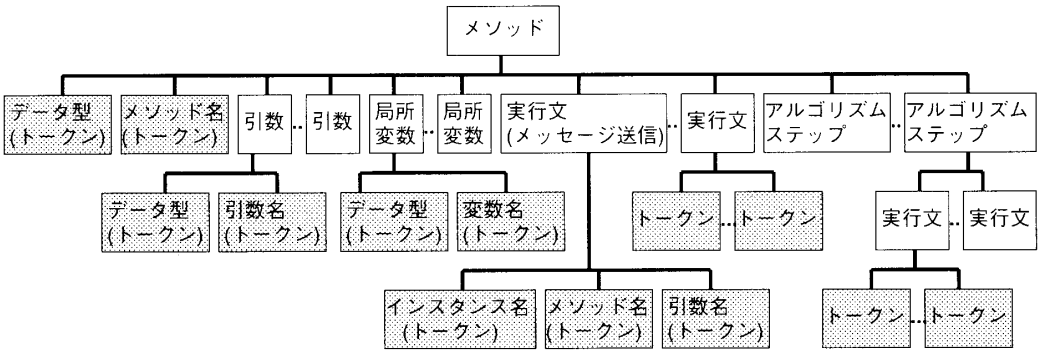

図 5 メソッドの理解過程モデル

プも同様である．以上より，メソッドの理解過程モデルは図 5 のようになる．

なお，文中にメッセージ送信が含まれる場合，文を理解する段階ではメッセージ送信のリンクが指すインスタンスが特定されていない．リンクによる関連を理解するためには OOP の構造を理解する必要がある．また，メッセージ送信によって OOP の状態は変化する．メッセージ送信による OOP の振舞いの理解過程については後述する．

ところで，基本チャンクには識別子や演算子，変数など様々な種類がある．これらの理解コストは，4.1 節の定義によりすべて 0 である．直観的には，基本チャンクの種類によって理解コストが異なると考えられる．しかし，我々が行った実験の結果，基本チャンクの種類による理解コストの違いは理解容易性の計量結果には大きな影響を与えない[14]．したがって，我々は基本チャンクの種類による理解コストの差を考慮しない．

クラス

クラスは，インスタンス変数，インスタンスの状態を変更するメソッド，クラス名，スーパークラス名によって定義される．これらの要素をソースコードに従って体制化すればクラスを理解できる．よって，クラスの理解過程モデルは図 6 のようになる．

しかし，クラスの理解だけでは OOP 全体を理解したことにはならない．静的な OOP はインスタンスの集まりで構成される†4ので，OOP を理解するために

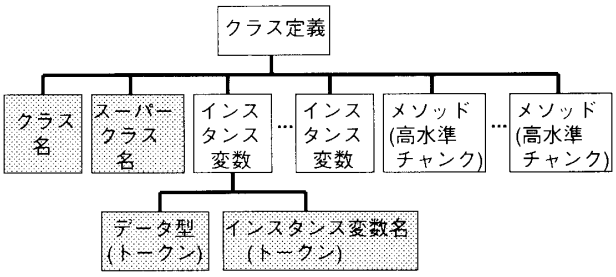

図 6 クラスの理解過程モデル

は各インスタンスの状態を理解する必要がある．インスタンスの状態を理解するためには，各インスタンスがどのようなメッセージをどのような順序で受信するかを把握する必要がある．これらを理解するためには，OOP の初期状態やイベントによる OOP の振舞いを理解しなければならない．

OOP の構造

静的な OOP の構造は OOP の初期状態によって理解される．これにより，個々のインスタンスが，どのクラスに所属し，リンクを通じてどのインスタンスと関連づけられているかを理解できる．以下に，OOP の初期状態の理解過程モデルについて説明する．

OOP の初期状態は各構成インスタンスの初期状態の組合せである．そのため，OOP の初期状態は各構成インスタンスの初期状態の集合を体制化することによって理解される．構成インスタンスの初期状態は，初期化イベントによりインスタンスが生成/初期化されることで設定される．インスタンスの生成/初期化はインスタンス定義に従って行われる．インスタンス定義はインスタンスが属するクラス定義とインスタ

†4 静的な OOP は実行中にインスタンス生成を行わないため，クラスを構成要素に含める必要はない．

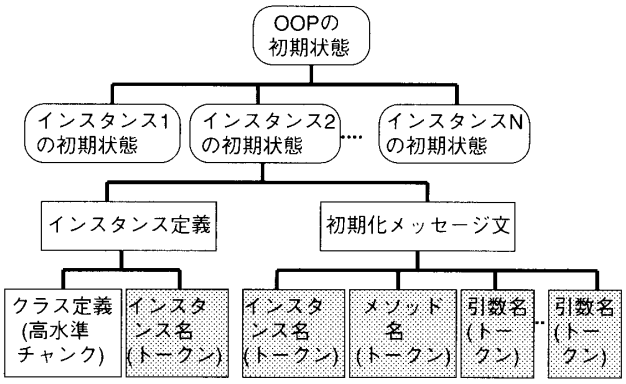

図 7 OOP の初期状態の理解過程モデル

ンス名から構成される．クラス定義は前述したクラスの理解過程モデルに従って体制化された高水準チャンクに対応する．以上により，OOP の初期状態の理解過程モデルは図 7 のようになる．

イベントによる OOP の振舞い

OOP の振舞いは，通常イベントに対応するイベントハンドラがインスタンスに対してメッセージを送信するたびに起こる．イベントハンドラがメッセージをインスタンスに送信すると，対応するインスタンスの状態変化が起こり，そのインスタンスから他のメッセージ送信が起こる．これに伴って，一般に複数のインスタンスで状態変化が起こる．したがって，OOP の振舞いを OOP に属する各インスタンスの状態変化の組合せによって定義する．

OOP を理解する際には，イベントごとに OOP の振舞いを理解する必要がある．これはイベントによってメッセージの到達範囲が異なったり，個々のインスタンスに到着するメッセージが異なるためである．本論文では OOP の振舞いを理解するために，各インスタンス間のメッセージ送信をトレースする．メッセージトレースによって各インスタンスの状態変化も理解できる．

以下，各インスタンスの状態変化の理解過程をモデル化する．インスタンス I に対するメッセージ m の送信を (m, I) で表す．この時のインスタンス I の状態変化を $s(m, I)$ で示す．(m, I) に伴ってインスタンス I が行うメッセージ送信を $(m_i, I_i)(1 \leq i \leq k, k$: I のメッセージ送信先のインスタンス数) で表す．異なるインスタンスへの同一メッセージ送信や同一インスタンスへの複数メッセージ送信を考慮すると，$i \neq j$ に対して $m_i \neq m_j, I_i \neq I_j$ とは限らない．また，メッセージ m を直接受信したインスタンス I とメッセージ m によって間接的にメッセージを受信したインスタンスからなる集合を，(m, I) に対する集合と定義する．この集合の (m, I) による振舞いを $S(m, I)$ で表す．つまり，$S(m, I)$ は (m, I) による OOP の振舞いに対応する．

メッセージ送信 (m, I) に対して，I が他のインスタンスへメッセージを送信しない場合を考える．この場合，振舞い $S(m, I)$ は m に対応するメソッドのみで理解できる．つまり，$S(m, I) = s(m, I)$ となる．インスタンスの状態変化 $s(m, I)$ は当該インスタンスが所属するクラスによって理解できる．クラスの理解は既に完了しているため，理解コスト $C(s(m, I)) = 0$ となる．

次に，(m, I) に対して I が他のインスタンス I_i にメッセージ m_i を送信する場合を考える $(1 \leq i \leq k)$. この場合，プログラマは (m, I) によって直接的および間接的にメッセージが送られるすべてのインスタンスの状態変化を理解しなければならない．インスタンスの状態変化を理解するためには，送信メッセージのトレースが必要となる．$S(m, I)$ は，$s(m, I)$ と $S(m_i, I_i)(1 \leq i \leq k)$ をチャンクとして体制化することにより理解できる．したがって，コスト式 (1) より，振舞いの理解コスト $C(S(m, I))$ を求めるコスト式 (3) が導ける．

$$C(S(m, I)) = C_R(k+1) + \sum_{i=1}^{k} C(S(m_i, I_i)) + 1 \quad (3)$$

式 (3) において $s(m, I)$ の理解コストは含まない．これは常に $C(s(m, I)) = 0$ となるためである．

OOP の振舞いに関係する一般的な概念として，多態性と動的束縛がある．定義した振舞いの理解過程モデルがこれらの概念に対応することを以下に示す．

インスタンス I, I' が継承によって関連づけられた異なるクラスに所属する場合，メッセージ送信 (m, I) と (m, I') によって起こる I, I' の振舞いには多態性がある．ここで，I, I' の所属クラスと m によって起

動されるメソッドの理解は，クラスやOOPの初期状態においてモデル化されている．したがって，振舞いの理解過程モデルは多態性の概念に対応している．

また，メッセージ送信 (m, I) に伴ってインスタンス I が行うメッセージ送信 $(m_i, I_i')(1 \leq i \leq k)$ の集合を理解することは，I が参照するインスタンス集合の動的束縛による振舞いを理解することと等価である．これらを理解するためには，I の所属クラス，m によって起動されるメソッド，リンクによって I に関連づけられたインスタンスを理解する必要がある．これらの理解は，すでにモデル化されたクラスやOOPの初期状態の理解に含まれている．

4.3 理解過程モデルの規則性

OOP内に同じ構造を持つインスタンスの部分集合(部分構造)が複数あるとする．各集合に同一のイベント(またはメッセージ)が送信された場合，各集合の振舞いが同一であれば，それらの集合には規則性がある．これらの各集合をそれぞれ単一の高水準チャンクとして体制化する際，最初に理解する集合の理解コストは式(1)等で計算されるが，他の集合の理解コストは0となる．これは，理解コストの定義より，同一チャンクに対する体制化は重複して行わないためである．

また，これらの集合の要素が同じクラスの異なるインスタンスであり，各集合が同一のイベント(またはメッセージ)が送信された場合も，各集合の振舞いが同一であれば規則性がある．よって，それらの集合の理解コストも重複して計算する必要はない．なぜなら，これらの集合に対する振舞いの理解過程が同一になるためである．

以上より，振舞いの理解過程モデル G_1, G_2 に以下の条件が成立する場合，規則性によって一方の理解コストを0とみなす．

(1) G_1 の節点集合 V_1 と G_2 の節点集合 V_2 の各要素が1対1で対応する．

(2) 1対1で対応する節点 $v = S(m, i) \in V_1$ と $v' = S(m', i') \in V_2$ について，$m = m'$ であり，i と i' が同一クラスに所属する．

(3) G_1 が枝 $e = (v_1, v_2)$ を含むとき，G_2 は枝 $e' = (v_3, v_4)$ を含む．また，G_2 が e' を含むとき，G_1 は e を含む．ここで，v_3, v_4 はそれぞれ v_1, v_2 に対応する節点である．

5 理解コストを用いた理解容易性評価

本節では，4節で定義した理解コスト計量法が，Chidamberメトリクスよりも的確に理解容易性を計量できることを示す．そのため，3.2節のChidamberメトリクスの検証と同様に，評価実験で使用した評価対象プログラムの理解容易性を理解コストで計算する．求めた理解コストと被験者の理解容易性の評価値平均に対して相関分析を行い，Chidamberメトリクスの場合と比較する．

評価実験において，各被験者の評価値を直接使わずに評価値平均を使っているのは，被験者の個人差による影響を可能な限り排除するためである．我々は文献[14]の研究を通じて，評価値に生じる個人差の主要な原因が被験者の知識レベルにあることを発見した．これに対して，本論文ではインスタンス数によるOOPの理解容易性への影響を主に議論するために，基本チャンクによって定義される一定の知識レベルに基づいた理解コスト計量法を構築している．そのため，理解コスト計量法に対する評価実験の際も評価値平均を利用している．

図8に，評価対象プログラムに対する理解コストと被験者の理解容易性の評価値の分布を示す．また，表3に被験者の評価値平均と理解コストの各値を示す．被験者が理解容易性を評価した値(評価値)と理

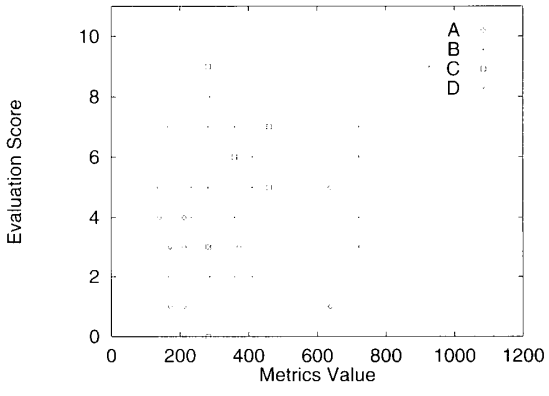

図8 理解コストと被験者の評価値の散布図

表 3 被験者の評価値平均と理解コストおよび両者の相関係数

基本プログラム名	インスタンス数	評価値平均	理解コスト	基本プログラム毎の相関係数
A 並列ソーティング	3	4.25	140	-0.33
	6	3.25	171	
	9	2.40	212	
	16	3.00	372	
	24	3.00	638	
B 並列行列計算	3	6.00	282	0.65
	5	4.33	411	
	7	5.00	721	
	11	9.00	927	
C ルーティング	8	4.80	282	0.82
	15	6.00	359	
	26	6.00	460	
D DNS	8	4.50	233	0.99
	16	4.80	287	
	28	5.25	359	
評価対象プログラムに対する評価値平均と理解コストの相関係数 (全体)				0.55

解コストとの相関係数は 0.55 であった．また，それらの順位相関係数は 0.72 であった．評価値平均と理解コストに対する基本プログラム別の相関係数では，基本プログラム B, C, D において高い相関があった．統計的検定を行った結果，評価値平均と理解コストが無相関であるとの仮説は有意水準 5% で棄却された．また，両者の間の母相関係数 ρ の 95% 信頼区間は $0.05 \leq \rho \leq 0.83$ である．

なお，プログラム毎の評価値平均と評価に要した時間の平均との相関係数は 0.49 であった．相関が低い理由は，プログラムが長いものでも，その内容が簡単であれば被験者は理解が容易であると回答したためである．

理解コストによる理解容易性評価について考察する．理解コストは，Chidamber メトリクスとは違って，インスタンス数により値が異なる．また，統計的検定の結果，Chidamber メトリクスは被験者の評価値とは無相関であったが，理解コストの場合は相関関係が認められた．これにより，インスタンス計量を考慮している理解コストは，Chidamber メトリクスよりも静的な OOP の理解容易性を的確に計量することができる．

2.4 節では，基本プログラム A(並列ソーティング) と基本プログラム B(並列行列計算) では，B の方が OOP の振舞いの理解が難しいと考察した．このことを理解コストでは正しく評価しているか，理解コストを詳細に分析する．基本プログラム A のクラスに対する理解コストは 85 であるのに対し，基本プログラム B のクラスに対する理解コストは 135 である．しかし，OOP の振舞いに対する理解コストは，インスタンス数 3 の場合，基本プログラム A の 55 に対して，基本プログラム B は 147 と約 4 倍になる．したがって，提案した理解コスト計量法を用いれば，基本プログラム B の理解が難しい理由は OOP の振舞いであることが示せる．

評価実験では，被験者が評価対象プログラムを理

表 4 評価値 10 を含めた被験者の評価値平均と理解コストおよび両者の相関係数

基本プログラム名	インスタンス数	評価値平均	理解コスト	基本プログラム毎の相関係数
A 並列ソーティング	3	4.25	140	-0.69
	6	3.25	171	
	9	2.40	212	
	16	3.67	372	
	24	3.00	638	
B 並列行列計算	3	8.00	282	0.70
	5	5.75	411	
	7	6.67	721	
	11	9.50	927	
	15	10.00	2730	
C ルーティング	8	4.80	282	0.82
	15	8.00	359	
	26	8.00	460	
D DNS	8	4.50	233	0.96
	16	4.80	287	
	28	6.20	359	
評価対象プログラムに対する評価値平均と理解コストの相関係数 (全体)				0.60

解不能 (評価値 10) と判断した場合や誤った理解をしている場合はその評価値を除外した．しかし，理解不能の場合，理解が難しいと被験者が判断していると考えれば，評価値として利用することも可能である．そこで，評価値 10 を含めた被験者の評価値と理解コストの相関係数も求めた．その結果，両者の相関係数は 0.60 であった．評価値 10 を含めた被験者の評価値平均値と理解コストとの基本プログラム毎の相関係数を表 4 に示す．また，統計的検定を行った結果，評価値平均と理解コストが無相関であるとの仮説は有意水準 5% で棄却された．また，両者の間の母相関係数 ρ の 95% 信頼区間は $0.15 \leq \rho \leq 0.85$ である．したがって，評価値 10 を含めた場合でも，被験者の評価値と理解コストの間には相関が認められた．

6 おわりに

本論文では，OOP の理解容易性を計量する際に，クラスだけでなくインスタンスの計量も必要であることを，被験者が行った理解容易性の評価実験や Chidamber メトリクス値と被験者の評価値の相関分析によって示した．

また，これらの分析をもとに，静的な OOP の理解過程をモデル化した．OOP を理解するには OOP の構造と個々のイベントによる OOP の振舞いを理解する必要がある．これらを理解するには個々のインスタンスの状態を理解しなければならない．この過程を系統的に行うために，各クラスを理解し，インスタンス間でのメッセージ送信をトレースする．静的な OOP の理解過程モデルを用いた理解コスト計量法では，クラスだけでなくインスタンスも考慮した計量が行える．

さらに，理解コストと被験者の理解容易性の評価値の相関分析を行った．その結果，Chidamber メトリクスよりも理解コストの方が理解容易性の度合いをより的確に導き出せることが示された．

ソフトウェアの理解度がプログラマに依存することは経験的に知られている．しかし，本論文では議論を単純化するためにこれを考慮していない．ただし，文献[14]において，(1) 理解度の属人性がプログラマの知識レベルに依存する，(2) 知識レベルは基本チャンクの違いにより説明できる，ことを示している．

2節の評価実験で使用した4種類の基本プログラムは，AとBのインスタンス図は常に一次元リスト，Cのインスタンス図は任意のグラフ構造，Dのインスタンス図は常に木構造となる．4.3節に述べたプログラム構造やメッセージ送信の規則性を考慮すると，理論的にはインスタンス数が増大した場合の理解コストの増加の割合は，AやBよりもDの方が，Dよりも Cの方が大きくなる．このことから，特定のインスタンス数 (プログラム構造) に依存しない形でOOPの理解容易性を評価するためには，理解コスト値 $C(n)$ とインスタンス数 n との関係を関数で示し，$C(n)$ のオーダーを評価することが有用と考えられる．

今後の課題としては，(1) 一般のOOPの理解コスト計量法に関する研究，(2) 保守コストの低減に関する研究，(3) 各種設計技術の定量的な比較に関する研究などが挙げられる．

謝辞

評価実験にご協力いただいた佐賀大学理工学部，九州大学大学院システム情報科学研究科，九州工業大学情報工学部の教官，学生の皆様に心から感謝いたします．

参考文献

[1] Chidamber, S. R. and Kemerer, C. F.: A metrics suite for object oriented design, *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6(1994), pp. 476–493.
[2] Henderson-Sellers, B.: *Object-Oriended Metrics: Measures of Complexity*, Prentice Hall, 1996.
[3] K. アウター: *XP* エクストリーム・プログラミング適用編—ビジネスで勝つための *XP*, ピアソンエデュケーション, 2002.
[4] Lieberherr, K. J. et al.: Object-oriented programming: an objective sense of style, in *Proc. OOPSLA*, 1988, pp. 323–334.
[5] Loftus, G. R. and Loftus, E. F.: *Human Memory: The Processing of Information*, Lawrence Erlbaum Associates, 1976.
[6] McClure, C.: *The Three Rs of Software Automation : Re-engineering, Repository, Reusability*, Prentice-Hall, 1992. (邦訳：ベスト CASE 研究グループ：ソフトウェア開発と保守の戦略，共立出版，1993.)
[7] McConnell, S.: *Code Complete*, Microsoft Press, 1993. (邦訳：石川：CODE COMPLETE —完全なプログラミングを目指して—，アスキー出版局, 1994.)
[8] Myers, G. J.: *Reliable Software Through Composite Design*, Mason/Charter, 1975. (邦訳: 久保, 国友: 高信頼性ソフトウェア—複合設計, 近代科学社, 1976.)
[9] Rumbaugh, J. et al.: *Object-Oriented Modeling and Design*, Prentice Hall, 1991. (邦訳：羽生田：オブジェクト指向方法論 OMT—モデル化と設計—，トッパン，1992.)
[10] Takehara, M., Kamiya, T., Kusumoto, S. and Inoue, K.: Empirical evaluation of method complexity for C++ program, *IEICE Trans. Inf. & Syst.*, Vol. E83-D, No. 8(2000), pp. 1698–1700.
[11] Yourdon, E. and Constantine, L. L.: *Structured Design*, Yourdon Press, 1975.
[12] 磯田: オブジェクト指向モデリング, コロナ社, 1998.
[13] 国友: セブン・イレブン流心理学, 三笠書房, 1999.
[14] 山崎, 松原, 掛下: 認知心理学的アプローチに基づくソフトウェア理解度計量法, コンピュータソフトウェア, Vol. 16, No. 6(1999), pp. 55–67.
[15] 御領他: 最新認知心理学への招待—心の働きとしくみを探る—, サイエンス社, 1993.
[16] 落水, 東田: オブジェクトモデリング, アジソン・ウェスレイ・パブリッシャーズ・ジャパン, 1998.