ソフトウェア工学 126−12
(2000. 3. 10)

静的なオブジェクト指向プログラムに対する理解コスト計量法と実験的評価

山崎 直子
有明工業高等専門学校
電子情報工学科
naoko@ariake-nct.ac.jp

掛下 哲郎
佐賀大学理工学部
知能情報システム学科
kake@is.saga-u.ac.jp

静的なOOPの理解過程をモデル化することで、その理解容易性を求める計量法を提案する。OOPを理解するにはプログラム構造と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。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。本計量法は我々が先に提案した理解コストを静的なOOPに適用したものである。本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。また、本計量法に対する実験的評価も行う。4種類中2種類のOOPの理解容易性に対し、被験者の評価値と本計量法で求めた計量値の間には高い相関がある。

An Understandability Metrics and Its Experimental Evaluation for Static Object Oriented Program

Naoko Yamasaki
Department of Electronics and Information Engineering
Ariake National College of Technology

Tetsuro Kakeshita
Department of Information Science
Saga University

We propose a method to calculate understandability for a static OOP through understanding process modeling. In order to understand an OOP, structure and behavior of the program must be understood for each event. A programmer then must understand state of each instance in order to understand program structure and behavior. Our understanding process model combines understanding of each class and message trace among instances. The proposed method applies our cognitive metrics to static OOPs. It can calculate understandability considering instances as well as classes. We evaluate the proposed method through an experiment. There are high correlations between the understandability scores by the examinees and the understanding costs for two OOPs among four.

1 はじめに

ソフトウェアライフサイクルにおいて約80%のコストが保守に費されている。大規模なソフトウェアになるほど保守の割合が大きくなる。また、保守コストの約50%は、ソフトウェアの内容を理解してバグの場所や変更箇所を特定することに費されている。従って、ライフサイクルコストを下げるためには、ソフトウェアの理解容易性を高めることが必要である。それには理解容易性の計量が必要である。

ソフトウェアメトリクスとは、ソフトウェアの品質や開発プロセスを定量的に評価するための手法である。オブジェクト指向に対応したメトリクスとして、Chidamber[1]らによるメトリクスが知られている。Chidamberメトリクスでは計量対象をクラスのみとし、分離度、継承木の深さ、クラスのレスポンス数などを計量する。しかし、我々が行った評価実験の結果、オブジェクト指向プログラムを理解するためにはクラスの計量だけでは不十分であることがわかった[2]。これは、プログラムのインスタンス構造やメッセージによるプログラムの状態変化の理解を考慮していないことに起因する。

我々は、これまで感覚的に論じられてきた理解容易性を定量的に求めるメトリクスとして理解コスト* を提案している[3]。我々は認知心理学で用いられる概念を活

* 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。

用して人間の理解過程をモデル化し、このモデルに基づき構造化プログラムの理解コストを定義した [4]。ここで、ソフトウェアの理解とは、当該ソフトウェアの仕様および内部構造を理解することである。理解コストは、認知心理学の成果を利用しているため理論的根拠が明確である。また、ソフトウェア開発上のあらゆる概念に対応できるため、設計や実装などの様々なレベルでの計量ができる。さらに、パラダイムの異なるソフトウェアの計量にも対応できる。

本稿では、静的なオブジェクト指向プログラム (OOP) を対象とした理解コスト計量法を提案する。静的な OOP とは、インスタンスとリンクからなるプログラム構造が実行中に変化しないものである。OOP を理解するためには、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。これらを理解するためには、個々のインスタンスの状態を理解する必要がある。この過程を系統的に行なうために、本稿ではプログラムの実行に沿った理解過程を採用し、インスタンス間でのメッセージ送信をトレースする。この際、(1) 同一メソッドは重複して理解する必要がない、(2) 複数イベントによる共有インスタンスの状態を理解する必要がある、(3) クラスを理解することで理解コストが低減できる、を考慮する。また、提案した理解コスト計量法に対する評価実験を行なう。実験では、オブジェクト指向の概念を理解している被験者に評価対象となる静的な OOP の理解容易性を評価させる。そして、被験者の評価値と理解コストの間の相関を求める。実験の結果、本稿で提案した理解コスト計量法は、静的な OOP の理解コストを求める際に有効であることが確認できた。

以下、2 節では理解コストの定義に必要な認知心理学の基本概念を説明する。3 節では理解コストの定義を示す。4 節では静的な OOP を定義し、それに関係する仮定を述べる。5 節では、静的な OOP を構成する各要素について、その理解過程をモデル化する。6 節では 5 節で定義した計量法の正当性を検証するための実験について説明する。7 節では実験結果と考察を述べる。8 節ではまとめと今後の課題を述べる。

2 認知心理学の基本的事項

認知心理学では、感覚貯蔵庫、短期貯蔵庫、長期貯蔵庫の 3 つによって人間の記憶装置をモデル化している [5]。感覚貯蔵庫は非言語的な情報を大量に蓄えるが、その内容は約 1 秒以内に消失する。感覚貯蔵庫の情報の一部が短期貯蔵庫に転送される。転送の過程でパターン認識が行なわれ、情報は言語的な形に変換される。

短期貯蔵庫はチャンクの体制化を行なう部分であり、人間の理解過程の中心的な役割を担う。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である。ここで、感覚貯蔵庫上での認識によって理解できるチャンクを基本チャンクと呼ぶ。また、体制化とは複数のチャンクからなるグループを一括理解し、それらをより高水準の単一チャンク (高水準チャンクと呼ぶ) にまとめる行為を指す。短期貯蔵庫内で行なわれるチャンクの体制化は高速に処理される。しかし、短期貯蔵庫の記憶容量は約 7 チャンクであるため、短期貯蔵庫に格納できないチャンクは長期貯蔵庫へ送られる。

長期貯蔵庫は、基本チャンクおよび体制化によって生成された高水準チャンクを単位として格納する記憶装置である。長期所蔵庫の処理は低速であが、その記憶容量は事実上無限大である。長期貯蔵庫に記憶されたチャンクは、将来の理解過程において再利用できる。

認知心理学の概念はソフトウェアの概念と対応する。チャンクは、トークン、文、変数、メソッド、インスタンス、クラスなど、ソフトウェアの様々な構成要素に柔軟に対応できる。また、体制化の過程はソフトウェアの理解過程に対応する。

このように認知心理学の概念を用いることで、ソフトウェアの構成要素の抽象度に依存しない汎用性の高いメトリクスを定義できる。また、異なるパラダイムに関する概念を全てチャンクとして統一的に扱えるため、パラダイムに対する独立性が高くなる。

3 理解コスト

ソフトウェアの理解とは、体制化を繰り返し行なうことによって、ソフトウェア全体に対応する単一の高水準チャンクを構成することである。我々は 1 回の体制化に要する理解コストを体制化に伴う長期貯蔵庫のアクセス回数で定義し、以下の式を導いた [3]。

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

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

$C(U)$ は n 個のチャンクからなるグループ (高水準チャンク)U の理解コスト、u_i はグループの構成要素である。また、$C_R(n)$ は n 個のチャンク間の全ての関連を理解するコスト (関連理解コスト) である。u_i が基本チャンクであれば $C(u_i) = 0$ となる。各 u_i の体制化が完了して

いるならば $C(u_i)$ は定数となる。体制化が完了していない場合には、u_i 自身を高水準チャンクと考えて再帰的に体制化する必要がある。そのためのコストを求める際にも式 (1) が適用できる。なお、同一チャンクに対する体制化は重複して行なわない。これは、一度体制化された高水準チャンクが長期貯蔵庫に記憶されるためである。

関連理解コスト $C_R(n)$ の値は、$n \leq 7$ のグループであれば線形にしか増加しないが、$n > 7$ のグループに対しては $O(n^2)$ で増加する。これは、チャンク間の関連を理解する際に、短期貯蔵庫の容量が 7 チャンクという制限によってチャンクの入れ替えが必要になるためである。

プログラムには、チャンク間に意味的な関連のないチャンクの集合が含まれることがある。例えば、チャンク間で共有変数がない、もしくは共有変数が参照のみに利用される場合である。本稿では、チャンクの順序を入れ替えても実行結果に変化のないチャンクの集合を無関連グループと定義する。無関連グループを理解する場合、チャンク数が 7 を超えてもチャンクの入れ替え操作は発生しない。従って、無関連グループの理解コストを以下のように定義する。

$$C_R(n) = n \qquad (2')$$

プログラムの理解コストを (1)、(2)、(2') 式によって計量するためには、プログラムの構成チャンクを決定する必要がある。一般に、プログラムの構成要素にチャンクを対応させる方法は複数存在する。例えば、関数は文を構成要素とする高水準チャンクであると考えられるが、明確な機能を持つ文の並び (アルゴリズムステップ) が空行などで明確になっていた場合、そのアルゴリズムステップを高水準チャンクと考えることもできる。また、人間は感覚貯蔵庫を用いることでプログラム中のパターンを低コストで高水準チャンクとして認識できる。このような場合には、理解コストが最小になる方法を選択する。これは、人間が経験的に理解しやすい方法を選択するためである。そのため、最小の理解コストはプログラムの本質的な複雑さと考えられる。構成チャンク数 n が 7 を越えるとその理解コストが $O(n^2)$ となる。従って、より多くの中間概念を設けることが理解コスト低減につながる場合が多い。

本稿では、上記の性質に基づき、理解過程のモデル化の際には理解コストがより低くなるように中間概念を適切に設定する。

4 静的な OOP

本節では、静的な OOP の定義とそれに関係する仮定を述べる。本節で述べる仮定は OOP の構成要素がチャンクの定義を満たすために必要である。一般のプログラムが本節で述べる仮定を守っているとは限らないが、この仮定を満たすように書き換えることはできる。従って、この仮定に基づいて OOP の理解過程モデルを作成しても、議論の一般性は失わなれない。

4.1 構造と振舞い

OOP は、大域的なインスタンス定義、メインプログラム、クラス定義、イベントハンドラから構成される。また、OOP では以下に示す 3 種類のイベントが発生する。

初期化イベント 初期化イベントはプログラム起動時に 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、メインプログラムを実行する。その後、プログラムはイベントループに入る。

通常イベント 通常イベントは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プログラム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プログラムは処理を行う。

終了イベント このイベントはプログラム終了時に起こる。このイベントにより、プログラム中のインスタンスを消去する。

4.2 静的な OOP の定義

静的な OOP は以下の 3 つの仮定を満たす。

- インスタンスは初期化イベントによってのみ生成される。
- リンクは初期化イベントによってのみ更新される。
- インスタンスは終了イベントによってのみ消去される。

上記の仮定により、プログラム構造は通常イベントによって変更されない。つまり、全ての通常イベントはインスタンスの生成/削除およびリンク値の変更を行えない。なお、通常イベントはインスタンスの状態を変更することはできる。

4.3 プログラミングスタイル

プログラムの理解過程モデルを作成する際には、プログラミングスタイルを指定する必要がある。これには2つの理由がある。1つはプログラムの構成要素がチャンクの定義を満たすようにするためである．もう1つは、プログラミングスタイルがプログラムの理解容易性に影響を与えるためである [4]。本稿では OMT[6] および文献 [7] のガイドラインに従って、プログラミングスタイルを以下のように指定する。

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

最後の仮定は、Demeter の法則 [8] に従うことを意味する。リンクはプログラムの実行中には変更されないので、各メッセージは送信先のインスタンスを静的に決定する。ゆえに、本稿では動的束縛、多態性を考慮しない。

5 OOP の理解過程モデル

静的な OOP は、プログラムの初期状態と、通常イベントが送られる毎に起こるプログラムの振舞いを体制化することによって理解される。静的な OOP の理解過程モデルを図1に示す。図1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、3節の定義に従って、各高水準チャンクの理解コストが計算できる。なお、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、丸型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。

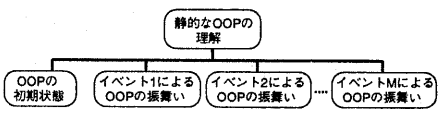

図 1: 静的な OOP の理解過程モデル

プログラムの初期状態や振舞いを理解するためには、まずソースコードの理解が必要である。本節では、ソースコードを理解するために必要なルーチンおよびクラスの理解過程をモデル化し、その後、OOP の初期状態と振舞いの理解モデルを構成する。

5.1 ルーチンの理解

ルーチンには、メインプログラム、サブルーチン、メソッドがある。ルーチンの理解過程は、構造化プログラムにおける場合と同一である。

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。3節の定義より、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一つの高水準チャンクとして理解される。入れ子の文に対しては、入れ子の内部から外部へ体制化を繰り返し行な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プも同様である。以上により、ルーチンの理解過程モデルは図2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。メッセージ送信のトレースが理解された時に、これらの関連も理解される。プログラムのメッセージ送信によって、プログラムの状態は次の状態へ変化する。メッセージ送信によるプログラムの振舞いの理解過程については 5.4 節に述べる。

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上記定義より同じ値である。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、我々が前回行なった実験の結果、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大きな影響を与えない [3]。よって、我々は基本チャンク毎の種類による理解コストの差を考慮しない。

5.2 クラスの理解

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、クラス名によって定義される。この中で、メソッドの理解過程モデルは 5.1 節で示した。これらの要素をソースコードに従って順に体制化すればクラスを理解できるので、クラスの理解過程モデルは図3のようになる。

しかし、クラスの理解だけではプログラム全体を理解したことにはならない。OOP はインスタンスの集まり

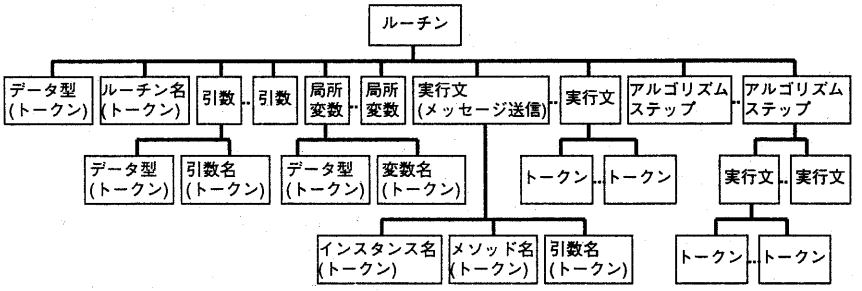

図 2: ルーチンの理解過程モデル

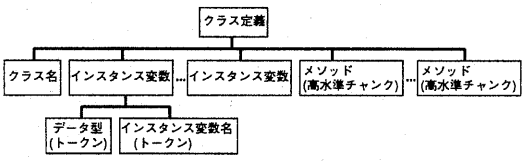

図 3: クラスの理解過程モデル

で構成される† ので、OOP を理解するために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。

5.3 OOP の初期状態

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なわれる。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ンス名から構成される。クラス定義は 5.2 節のクラスの理解によ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 4 のようになる。これにより、OOP のプログラム構造 (インスタンス構造) と全てのインスタンス間のリンク構造を理解できる。

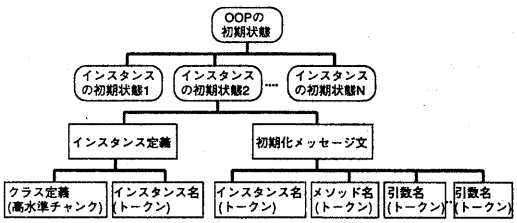

図 4: OOP の初期状態の理解過程モデル

5.4 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。従って、通常イベント毎の 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。

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 $(m_i, I_i)(1 \leq i \leq k)$ 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

† インスタンスの動的生成を行なわないため、クラスを構成要素に含める必要はない。

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 G を、(m,I) に対する集合と定義する。集合 G の (m,I) による振舞いを $S(m,I)$ で表す。

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。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全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。従って、理解コスト $C(S(m,I))$ は (3) 式によって求められる。

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

(3) 式において $s(m,I)$ の理解コストは含まれていない。これは常に $C(s(m,I))=0$ となるためである。

同一メッセージによる同一集合の振舞いを理解する場合、その振舞いを重複して理解する必要はない。また、この集合の要素が同じクラスの異なるインスタンスであっても同様である。なぜならば、これらの集合の振舞いの理解は各インスタンスの状態に依存しないからである。ゆえに、これらの集合の振舞いの理解コストは2回目以降0となる。

6 実験的評価

本稿では、5節で定義したOOPの理解過程モデルによる計量法の正当性を実験によって検証する。実験には、被験者と評価の対象となるプログラムが必要である。次に被験者と評価対象プログラムについて述べる。

実験の被験者は14名である。被験者の内訳は、大学の教官・技官3名、大学院生5名、学部生6人である。被験者は全て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、4種類のプログラムを基本とする。この基本プログラムはクラスのみが記述されている。基本プログラムの内容は以下の通りである。

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

B：1次元メッシュを用いてN×N行列とN次元ベクトルの乗算を行う並列行列計算プログラム

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

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)

4種類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3個の主プログラムをそれぞれ用意する。従って、全12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記述されているのではなく、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の種類が異なるためである。また、クラス名やメソッド名、ルーチン名にはその機能に関連のない名前が使用されている。インスタンス変数名のみがその意味を表す名前になっている。

評価対象プログラムにおいて、主プログラムのサブルーチン1は初期化イベントの並びであり、サブルーチン2は通常イベントの並びを示す。

評価実験は以下の手順で行なう。

1. 12種類の評価対象プログラムの理解コストを計量する。
2. 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。
3. 被験者は、各プログラムの理解容易性を10段階で相対的に評価する。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に、以下の4つの問いに答えさせる。
   (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は何か説明せよ。
   (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。
4. 理解コストと被験者による評価値の相関関係を調べる。

評価対象プログラムの理解コストを求めるためには、基本チャンクを決定する必要がある。ここで、基本チャンクは文とする。評価対象プログラムはアルゴリズムを

表 1: 被験者の評価値平均と理解コストおよびそれらの相関

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
A 並列ソーティング	3 5 7	4.25 3.25 2.00	138 184 242	-1.00
B 並列行列計算	3 6 9	8.00 5.75 6.50	250 388 673	-0.49
C ルーティング	8 15 26	3.75 7.33 7.33	426 644 872	0.86
D DNS	8 16 28	4.33 4.00 5.75	286 445 666	0.82
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			0.58	

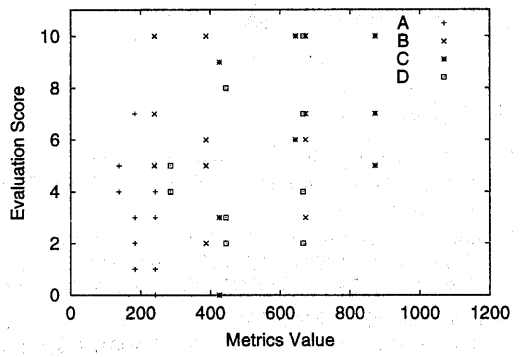

図 5: 理解コストと被験者の評価値の分布

自然言語で書いてあるため、各文を読むことで理解ができるからである。

さらに、基本プログラム A,B については、実行結果が何を示しているのか理解する必要がある。よって、実行結果を体制化することにより、理解コストの値を補正する。また、基本プログラム C,D については、主プログラムよりパターンが容易に認識できるため、これを高水準チャンクとする。また、高水準チャンクは無関連グループの定義を満たすので、体制化の際には式 (2') を用いて理解コストの値を補正する。

7 実験結果と考察

図 5 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 1 に被験者の評価値平均と理解コストの各値を示す。実験の結果、被験者が理解容易性を評価した値と理解コストとの相関は 0.58 であった。また、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では、基本プログラム C,D に高い相関がある。

表 1 より、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影響されていることがわかる。従って、OOP の理解容易性を求めるにはインスタンスの計量が必要である。

本計量法と Chidamber らが提案したメトリクス [1] を比較する。Chidamber メトリクスでは以下の計量を行う。

WMC クラス毎の重み付きメソッド数

CBO クラス間の結合度

RFC 評価対象クラスのメソッド数とそれらのメソッドによって呼び出されるメソッド数の和

LCOM クラス毎のメソッドの凝集性の欠如

この他に、Chidamber らは DIT(継承木の深さ) と NOC(クラス毎のサブクラス数) も提案している。しかし、実験で用いる評価対象プログラムのどのクラスもサブクラスを持たないので実験の対象とはしない。

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を表 2 に示す. Chidamber メトリクスはクラスを計量対象としているため、インスタンス数によるメトリクス値の変化はない。

表 2 より、メトリクス値と被験者の評価値平均の相関は低い。これは Chidamber メトリクスがクラスのみを計量対象とするためである。理解コスト計量法はクラス計

表 2: Chidamber メトリクス値と評価値平均との相関

基本プログラム名	WMC	CBO	RFC	LCOM
A	24	0	4	0
B	26	0	8	16
C	50	4	19	4
D	36	2	14	0
相関	0.30	-0.00	0.20	0.49

量だけでなくインスタンス計量も行うため、インスタンス数の違いに対応した理解容易性の違いを説明できる。

表 1 より、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では基本プログラム A,B の値が低い。つまり、インスタンス数が少ないほど評価値が高い。これは、インスタンス数が少ないため、インスタンスの状態変化から規則性が導きにくく、プログラムの機能の理解が困難なことに原因があると考えている。また、基本プログラム A,B の実行結果は両方ともいくつかの数字で出力される。出力データの個数はインスタンス数の少ないプログラムの方が少ない。そのため、出力データの意味を考える際に規則性が導きにくいことも原因として挙げられる。

これらの考察から、何の機能を持つか不明な OOP では、インスタンス数をある程度増やすことで規則性を導きやすくなると考えられる。一方、インスタンス数を増やすとプログラムの相互作用は複雑になるため、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。

8 おわりに

本稿では、理解コストを用いた静的な OOP の理解計量法を提案した。本計量法は静的な OOP に対する理解過程をモデル化することで求められる。また、提案した理解コスト計量法の正当性を検証するために、実験による評価を行った。それにより、4 種類の基本プログラム中、2 種類に対しては高い相関があった。

本計量法の特徴は、プログラム構造やその振舞いを考慮し、インスタンス単位での計量を行う点にある。プログラムの初期状態を理解するコストや通常イベントによってメッセージが送信された時のプログラムの状態変化を理解するコストは、クラスの理解以外にも必要なコストである。既存の OOP メトリクスのほとんどはクラスを計量対象としている。しかし、クラスの理解だけではプログラムのインスタンス構造とメッセージによるプログラムの状態変化を理解できない。従って、クラスを理解しただけではプログラム全体を理解したとはいえない。

理解コストは、ソフトウェア開発上のあらゆる概念に対応でき、パラダイムの異なるソフトウェアの計量もできる。そこで、我々は、この理解コストを、新しい設計技法を生み出す理論的基礎としても位置づけている。今後の課題としては、提案した計量法では挙げることのできなかった OOP の理解に必要な要素の考察と計量法の改良、動的な OOP への計量法の拡張などが挙げられる。

謝辞

被験者として協力していただいた、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皆様に深く感謝いたします。

参考文献

[1] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Engineering*, 20(6):476–493, 1994.

[2] 山崎, 掛下. Chidamber メトリクスを用いたオブジェクト指向プログラムの理解容易性評価. In **平成 *11* 年度 電気関係学会九州支部連合大会講演論文集**, page 763, Oct 1999.

[3] 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. **日本ソフトウェア科学会学会誌「コンピュータソフトウェア」**, 16(6):55–67, Nov 1999.

[4] 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の理解度計量法に対する実験的評価. In **ソフトウェア工学の基礎** *IV (FOSE '97)*, pages 59–66. 日本ソフトウェア科学会, 近代科学社, 1997.

[5] G. R. Loftus and E. F. Loftus. *Human Memory: The Processing of Information.* Lawrence Erlbaum Associates, 1976.

[6] James Rumbaugh et al. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[7] S. McConnell. *CODE COMPLETE.* Microsoft Press, 1993.

[8] K. J. Lieberherr et al. Object-oriented programming:an objective sense of style. In *Proc. OOPSLA*, pages 323–334, 1988.