ソフトウェア工学 108－7
（1996．3．22）

情報隠蔽に注目したオブジェクト指向設計の評価

保坂 大輔　　高田 義広　　鳥居 宏次

奈良先端科学技術大学院大学 情報科学研究科
〒630-01 奈良県 生駒市 高山町 8916-5
E-mail: (daisuk-h|takada|torii)@is.aist-nara.ac.jp

オブジェクト指向ソフトウェア開発において，ソフトウェアの変更容易性を評価するための指標となるソフトウェア尺度を提案する．提案する尺度は，変更容易性に強く関連していると思われる情報隠蔽に注目して定義されており，その情報隠蔽がソフトウェア内で効果的に行われている度合を表す．この度合を情報局所性と呼ぶ．尺度の定義に際しては，計測理論に基づく方法を採った．つまり，計測の目的を変更容易性の評価と定めて，変更容易性と情報局所性とその尺度との関係をできるだけ明確に述べるように注意した．提案する尺度を小規模なソフトウェアに適用したところ，経験的な評価と尺度の値とがよく合致した．

Evaluation of Object-Oriented Designs Focusing on Infomation Hiding

Daisuke Hosaka　　Yoshihiro Takada　　Koji Torii

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama-Cho, Ikoma-Shi, Nara 630-01, Japan

This paper presents a software metric indicates the ease with which object-oriented software can be modified. The metric focuses on information hiding, which is strongly associated with ease of modification, by indicating its extent. The extent is called "information locality." When we defined the metric, we took an approach based on measurement theory. That is, we first set the purpose of measument, and then clarified relations among ease of modification, infomation locality, and the metric. The metric was applied to a small example. The result agreed well with an evaluation based on experience.

1 はじめに

オブジェクト指向のソフトウェア開発方法論は，ソフトウェアの変更や部品化を容易にすることに有効であると言われている [2, 4, 7, 8]．しかし，オブジェクト指向方法論を適用しても，変更や部品化が常に容易になるとは言えない．その理由は，次の通りである．一般に，オブジェクト指向方法論では，そのようなソフトウェア品質を，ソフトウェアの評価と改良とを繰り返すことにより，段階的に洗練する [2]．ところが，オブジェクト指向ソフトウェアのそのような評価については，明確な指標が未だ確立されていない．そのために，オブジェクト指向開発の経験が少ない技術者にとっては，適正な評価を行うことが困難である．また，経験を積んだ技術者であっても，常に適正な評価を行えるとは限らない．従って，方法論の通りに開発を進めたとしても，期待する品質が得られない危険性がある．このような評価指標の欠如は，技術者にとってオブジェクト指向方法論の習得を困難にしている原因の１つであるとも考えられる．

そこで，本稿では，オブジェクト指向ソフトウェアの変更容易性を評価するための新たな尺度を提案する．この尺度は，ソフトウェア内で情報隠蔽が効果的に行われている度合を意味する．ここで言う情報隠蔽とは，オブジェクト指向ソフトウェアの変更容易性を高めるための基本的な技術である．詳しくは3章で述べるが，ソフトウェア中で変更が起こった時に，その影響の波及する範囲をできるだけ小さくする技術である．この度合を，以降，情報局所性と呼ぶことにする．

オブジェクト指向ソフトウェアの評価については，従来より多くの尺度が提案されているが [3]，実用化されている尺度はほとんどない．その原因としては，オブジェクト指向方法論の普及が最近であるだけでなく，従来の研究の方法にも問題があったと考えられる．従来の尺度の多くは，使用する目的を明確にしないまま定義され，その後に，ソフトウェア品質との関連が実験データを使って統計的に分析されていた．つまり，実際に評価の要求されているソフトウェア品質とその尺度との間の関係が，必ずしも論理的に説明されていなかった．従って，統計的分析の結果，尺度の値とソフトウェア品質との間に関連が見られるとは限らなかったし，見られたとしても，他の条件の下で同様の関連が見られるかどうかを追究することが困難であった．

そこで，情報局所性の尺度を定義するに際しては，まず，計測の目的を変更容易性の評価と定めて，変更容易性と情報局所性とその尺度との関係をできるだけ明確に述べるように注意した．このような方法により，より目的に合う尺度が得られることが期待される．また，尺度の有効な範囲が明確になるし，尺度の改良が容易になると考えられる．尺度の定義におけるこのような目的指向の重要性については，文献 [5, 6] などで紹介されているソフトウェアの計測理論 (measurement theory) や，文献 [1] に述べられている GQM Paradigm においても強調されている．

情報局所性の尺度を定義した具体的な手順は次の通りである．まず，オブジェクト指向ソフトウェアの情報のモデルを定義して，できるだけ厳密に情報隠蔽を定義し，変更容易性と情報隠蔽との関係を述べる．(3章)．次に，その情報モデルに基づいて，計測する性質である情報局所性を定義し，同時に，それを数値への写像する方法，つまり，尺度を定義する．(4章, 5章)．尺度の定義は，2 段階に分けて行った．まず，「モジュールの持つ情報」と言う言葉の解釈によって意味の変わる抽象的な情報局所性の定義を与えた (4章)．隠蔽の対象である情報については，現在も研究中であり，将来的に拡張した定義を与える可能性があるからである．次に，情報に対して現時点で妥当と思われる定義の１つを与え，具体的な情報局所性の尺度を定義した (5章)．この尺度を小規模なプログラムに適用したところ，経験的な評価と尺度の値とがよく合致した (6章)．

2 オブジェクト指向ソフトウェア

情報のモデルについて述べる前に，本稿で用いる基本的な用語について整理しておく．

まず，ここで言うオブジェクト指向ソフトウェアとは，協調して働く多数のオブジェクトから構成されるシステムの動作を記述したプログラム，または，詳細な設計文書である．言語や方法論によっては，オブジェクトとその他の数値とを区別している場合と，区別せずに全ての存在をオブジェクトであるとみなしている場合とがある．ここでは，便宜的に，全ての存在をオブジェクトとみなす．同じ構造を持つオブジェクトの集まりを**クラス**と呼ぶ．

各オブジェクトは，そのオブジェクトの状態を表す 0

個以上の属性を持つ．ここでは，数値もオブジェクトと考えるので，各属性もオブジェクトである．そこで，各オブジェクトは，0個以上のオブジェクトへのポインタを持つと考え，それらのポインタをそのオブジェクトの**属性**と呼ぶことにする．属性の個数や各属性のクラスは，属性を持つオブジェクトのクラスにより定まる．

オブジェクト間では，メッセージが送受信される．メッセージを受信したオブジェクトは，その種類に応じた手続き，つまり，**メソッド**を起動する．メッセージを送信してメソッドを起動することを，メソッドを呼び出すと言う．オブジェクトの持つメソッドの個数や種類は，そのオブジェクトのクラスにより定まる．

メソッドを呼び出す時には，0個以上のオブジェクトを引数として指定できる．そして，メソッドが終了すると，1，または，0個のオブジェクトが戻り値として返される．引数や戻り値の個数やクラスは，メソッドより一定である．

オブジェクトの持つ属性については，言語や方法論により，他のクラスのオブジェクトから直接的に参照できる場合とできない場合とがある．ここでは，オブジェクトがその属性を読み出すためのメソッドを持っていて，そのメソッドを呼び出した時にだけ，他のオブジェクトがその属性を参照できることにする．

クラスには階層関係が存在する．あるクラスに属するオブジェクトは，その上位のクラスにも同時に属する．上位のクラスのオブジェクトが持つ種類の属性やメソッドは，下位のクラスに**継承**される．

オブジェクトの中には，システムの動作中に恒久的に存在するオブジェクトと，生成されたり消去されたりするオブジェクトがある．恒久的に存在するオブジェクトを**静的オブジェクト**と呼ぶ．

オブジェクト指向ソフトウェアの開発においては，一般に，クラスごとに定義が作成される．そして，各クラスの定義が作成される時には，そのクラスに関連する静的オブジェクトの定義も合わせて作成される．そこで，あるクラスの定義と関連する静的オブジェクトの定義とを，**モジュール**と呼ぶことにする．

モジュールの作成においては，一般に，まず，各モジュールに対して，他のモジュールとの境界において要求される表1のような項目が決定される．次に，その要求に従ってモジュールが作成される．そのような要求を**仕様**と呼び，仕様に従う作成を**実装**と呼ぶ．

表1: モジュールの仕様の項目

そのモジュールに対応するクラスの識別子
そのクラスのオブジェクトの各属性の形式
そのクラスのオブジェクトの各メソッドの形式
そのクラスのオブジェクトの各メソッドの意味
そのクラスに関連する各静的オブジェクトの形式

メソッドの形式とは，メソッドの識別子，引数の個数，各引数のクラス，戻り値のクラスなどを列挙したものである．それに対して，メソッドが呼び出された時のオブジェクトの動作の仕様を，メソッドの意味と呼ぶ．

仕様には，オブジェクトの属性に対する要求も含まれる．仕様には，各メソッドの意味が含まれ，しかも，メソッドの多くはオブジェクトの属性を変化させるからである．ただし，モジュールを作成する時には，必ずしも要求通りの属性を定義しなくてよい．前述のように，オブジェクトの属性は，メソッドの呼び出しによってだけ参照したり変更できる．それらのメソッドに工夫をすれば，あたかもその属性が存在するかのように，そのオブジェクトの別の属性，あるいは，他のオブジェクトの属性を使って見せかけることが可能である．仕様に表れる属性を**仮属性**と呼び，オブジェクトが実際に持つ属性を**実属性**と呼んで，区別することにする．

3 情報のモデル

変更容易性と情報隠蔽との関係を明確にするために，オブジェクト指向ソフトウェアの情報モデルを定義する．このモデルは，以下の7個の概念の定義からなる．

3.1 モジュールの持つ情報

まず，隠蔽の対象である「情報」について述べる．ソフトウェアやモジュールは，特定の言語や記法で情報を表現したものであると考えられる．ただし，隠蔽の対象は，モジュールとして表現される情報だけでない．例えば，あるメソッドの意味は，隠蔽の対象であるが，多くの言語では，メソッドの形式を記述するが，意味を記述しない．隠蔽の対象になる情報とは，モジュールとして表現される情報を含めて，モジュールが作成される時に参照される情報である．

ただし，開発時に参照された情報であっても，開発途中で必要なくなれば，隠蔽の対象にならない．結局，各モジュールについて，隠蔽の対象となる情報とは，その

モジュールの開発のために必要な最低限の情報であると言える．これを，**そのモジュールの持つ情報**と呼ぶことにする．同様に，ソフトウェアの開発のために必要な最低限の情報を，そのソフトウェアの持つ情報と呼ぶ．モジュール m の持つ情報の集合を $I(m)$ と表すことにする．もし，$I_1 \subseteq I(m)$ ならば，モジュール m が情報 I_1 を持つと言うことにする．

具体的に言うと，$I(m)$ とは，m の仕様，m の使用する他のモジュールの仕様，その他の実装に関する情報などである．言語や方法論やプログラミング技術などは，隠蔽の対象とならないので，モジュールの持つ情報に含めない．情報の具体的な定義については，5章で述べる．本モデルでは，抽象的な定義を与えるに止める．

この定義に従えば，同一の情報が複数のモジュールに重複して持たれることがある．例えば，あるクラスのオブジェクトのメソッドが他のクラスのオブジェクトから呼び出されている場合，両クラスに対応する 2 個のモジュールは，呼び出される側のクラスの仕様を共通に持つ．

3.2 情報量

本モデルでは，情報についての具体的な定義を与えないので，情報量についても，その満たすべき性質だけを挙げて，抽象的に定義する．情報 I_1 の量を $q(I_1)$ と表すことにしよう．

第 1 に，$q(I_1)$ は，非負実数である．負の量を持つ情報は，考えられないからである．

第 2 に，$q(I_1)$ は，I_1 の表現だけの相違に影響されない量である．同一のモジュール m については，C++言語で表されていても，設計文書で表されていても，それが持つ情報の量 $q(I(m))$ が同一である．

第 3 に，情報 I_1 は，それ以上に分割できない単位要素の集合であり，$q(I_1)$ は，I_1 に含まれる単位要素の量の和 $\sum_{i \in I_1} q(i)$ であると考えることにする．

3.3 モジュールの規模

まず，1 個のモジュールにおける規模と情報量との関係について述べる．モジュールの持つ情報が多い程，そのモジュールは，作成や変更に多くの労力が必要となり，大きいと考えられる．そこで，モジュールの持つ情報の総量を，そのモジュールの**規模**と呼ぶことにする．モジュール m の規模を $c_{\langle I,q \rangle}(m)$ とすると，m の持つ情報が $I(m)$ であるので，次式が成り立つ．

$$c_{\langle I,q \rangle}(m) = q(I(m))$$

次に，モジュールの集合 $M = \{m_1, m_2, \ldots, m_n\}$ の規模 $c_{\langle I,q \rangle}(M)$ について述べる．これには，次の 2 通りの定義が考えられる．

$$c_{\langle I,q \rangle}(M) = q\left(\bigcup_j I(m_j)\right)$$

$$c_{\langle I,q \rangle}(M) = \sum_j q(I(m_j))$$

前の定義では，例えば，類似した 10 個のモジュールの規模は，1 個のモジュールとほぼ同じであると仮定されていて，後ろの定義では，1 個のモジュールの 10 倍であると仮定されている．ここでは，後の仮定を選び，ソフトウェアの規模を，モジュールの規模の和とみなす．

3.4 ソフトウェアの変更

実際のソフトウェア開発では，修正，移植，拡張，性能改良などのための多様な変更が頻繁に起こる．ここでは，そのような変更を，モジュールの持つ情報の追加，変更，削除に対応付ける．

まず，ソフトウェアの変更は，ソフトウェアの持つ何らかの情報 I_0 の変更から始まると考える．情報は複雑に関連しているので，一部が変更されると，他の部分を追加したり変更したり削除したりする必要が生じる．そのような追加，変更，削除などの操作がさらに同様の操作を起こすこともある．そして，それ以上の操作が必要なくなるまで操作が繰り返される．また，並行して，情報の操作を反映するようにモジュールとしての表現も書き変えられる．以上の操作を，I_0 についての**ソフトウェアの変更**と呼ぶことにする．

3.5 変更容易性

情報 I_0 についてのソフトウェアの変更に要する労力は，さまざまな要因に影響される．例えば，情報 I_0 の性質，ソフトウェアの性質，変更する人間の性質などである．それらの中のソフトウェアの性質を，I_0 についての**ソフトウェアの変更容易性**と言う．

変更の定義に従えば，この変更容易性は，実際の変更を行わない限り計測できない性質である．変更は，情報の操作からなるので，変更の労力は，操作の加えられた情報，または，操作のために参照された情報の量と密接

な関連を持つ．ところが，任意の情報I_0についてそのような情報の量を予測する方法は得られていない．

3.6 準変更容易性

実際にソフトウェアを変更しなくても簡単に計測できて，しかも，情報I_0についての変更容易性と密接な関連を持つ性質について述べる．そのようなソフトウェアの性質を，I_0についてのソフトウェアの**準変更容易性**と呼ぶことにする．

前述のように，I_0についての変更のために参照する必要のある情報は，変更の労力と密接な関連を持つが，容易に特定できない．そこで，I_0と共通の情報を持つモジュールの集合$\{m|I(m)\cap I_0\neq\phi\}$を$M(I_0)$と表すことにする．$M(I_0)$の持つ情報は，情報や情報量の解釈$\langle I,q\rangle$にもよるが，比較的容易に特定できる．しかも，その規模$c_{\langle I,q\rangle}(M(I_0))$は，変更の労力と密接な関連を持つと考えられる．なぜなら，$M(I_0)$の持つ情報は，I_0と関連があるかもしれないので，I_0についての変更の時に少なくとも1度は参照されるからである．従って，$c_{\langle I,q\rangle}(M(I_0))$の小ささは，$I_0$についての準変更容易性の1つであると言える．

3.7 情報隠蔽

ソフトウェアの変更容易性と情報隠蔽との関係について述べる．情報I_0についてのソフトウェアの変更容易性は，前述のように，$c_{\langle I,q\rangle}(M(I_0))$の小ささと関連がある．$c_{\langle I,q\rangle}(M(I_0))$を小さくするには，可能な限り，$M(I_0)$に含まれるモジュールの数が少なく，そして，それぞれのモジュールの規模が低くなるようにモジュールを作成すればよい．つまり，I_0と関連するモジュールの集合をできるだけ小さくして，I_0を他のモジュールから隠蔽すればよい．逆に，そのように$M(I_0)$を小さくしてI_0を隠蔽すれば，$c_{\langle I,q\rangle}(M(I_0))$が小さくなり，変更容易性が高くなることが期待される．そこで，上述のように$c_{\langle I,q\rangle}(M(I_0))$を小さくすることを，$I_0$についての**情報隠蔽**と呼ぶことにする．

4 抽象的な情報局所性

情報モデルに基づいて，情報隠蔽が効果的に行われている度合を表す情報局所性を定義する．ただし，情報モデルでは，情報や情報量について抽象的な定義しか与えていない．そこで，本章でも，情報局所性に対して抽象的な定義，つまり，情報や情報量の解釈$\langle I,q\rangle$によって意味の変わる定義を与える．これを，抽象的情報局所性$h_{\langle I,q\rangle}$と呼ぶことにする．

情報モデルでは，情報I_0についてのソフトウェアの変更容易性を高めるために，$c_{\langle I,q\rangle}(M(I_0))$を小さくすることを，$I_0$についての情報隠蔽と定義した．従って，情報隠蔽が効果的に行われていることとは，変更の要求される可能性のあるI_0について$c_{\langle I,q\rangle}(M(I_0))$の期待値が小さいことであると考えられる．仮に，そのような期待値が小さいことを情報局所性とみなし，I_0の変更が要求される確率を$\Pr(I_0)$と表すことにすれば，次のような定義式が得られる．

$$h_{\langle I,q\rangle}=\left[\sum_{I_0}\Pr(I_0)\cdot c_{\langle I,q\rangle}(M(I_0))\right]^{-1} \tag{1}$$

ただし，$\Pr(I_0)$は，情報の解釈Iにもよるが，状況に依存する分布であり，推定が困難である．そこで，情報I_0が，それ以上に分割できない単位要素であり，かつ，変更の要求される可能性がある場合は，$\Pr(I_0)$が情報量に比例する値$\alpha\cdot q(I_0)$である（αは比例定数）と仮定し，そうでない場合は，$\Pr(I_0)=0$であると仮定することにする．変更の可能性のある単位要素の情報の集合をEと表すことにすると，$\sum\Pr(I_0)=1$であることから，$\alpha=\left[\sum_{i\in E}q(i)\right]^{-1}$であることがわかる．以上の仮定に基づくと，定義式(1)を次のように書き直せる．

$$h_{\langle I,q\rangle}=\frac{\sum_{i\in E}q(i)}{\sum_{i\in E}\left(q(i)\sum_{m\in\{m|i\in I(m)\}}q(I(m))\right)} \tag{2}$$

これを最終的に抽象的情報局所性と定義することにする．つまり，抽象的情報局所性$h_{\langle I,q\rangle}$とは，$\Pr(I_0)$についての仮定の下での，I_0に関連するモジュールの規模の和の期待値の小ささを意味する．

5 具体的な情報局所性の尺度

前章で述べた抽象的情報局所性を詳細化し，具体的な情報局所性尺度を定義する．つまり，$h_{\langle I,q\rangle}$の定義式(2)において，モジュールmの持つ情報$I(m)$と，情報I_1の量$q(I_1)$とに具体的な定義を与えることにより，測定が可能な程度に具体的な$h_{\langle I,q\rangle}$の定義を得る．前述のように，モジュールの持つ情報や情報量については，現在も研究中であるので，ここでの定義は，現時点で妥当と思われる解釈の1つである．抽象的な定義とそのような解釈とを区別するために，本章で定義する

情報や情報量を $\langle I'(m), q'(I_1)\rangle$ と表し，そして，得られる情報局所性尺度を $h_{\langle I',q'\rangle}$ と表すことにする．

5.1 情報の分類

モジュールの持つ情報に具体的な定義を与える前に，そのような情報を構成する要素の分類を与えておく．2章で述べたように，一般に，モジュールは仕様に従って実装される．そして，3.1で述べたように，モジュール m を作成するために必要な情報は，m の仕様，m の使用する他のモジュールの仕様，その他の実装に関する情報に分類される．

それらの3種類の情報の内，仕様に関する2種類については，さらに表1のように分類できる．実装に関するその他の情報としては，そのクラスのオブジェクトの持つ実属性の形式，そのクラスのオブジェクトの持つメソッドの中で使われるアルゴリズム，そのクラスの上位クラスが挙げられる．

5.2 情報の具体的な定義

前節で分類した情報の要素の内，メソッドの意味やアルゴリズムなどについては，現時点において形式化が困難である．そこで，ここでは，表2のように，現時点で形式化できる情報だけを，近似的に，モジュールの持つ情報とみなすことにする．

まず，クラスの識別子，メソッドの形式，属性の形式，静的オブジェクトの形式などは，全てモジュールの持つ情報に含めることにする．そして，メソッドの意味やメソッドの中で使用されるアルゴリズムなどについては，各メソッド内で参照している実属性，静的オブジェクトや，呼び出しているメソッドだけを，モジュールの持つ情報に含めることにする．具体例は6章で示す．

5.3 情報の形式的な表現

情報を形式的に表すことは，情報量の具体的な定義を考える上で重要である．ここでは，前節で述べた情報の各要素を，一階述語論理の枠組を利用して，表3のように形式的に表すことにする．具体例は6章で示す．

一階述語論理を採用した理由としては，枠組が単純であり意味が明快であること，および，多様な情報を表現できる能力があることが挙げられる．特に，将来的に，モジュールの持つ情報の定義を，メソッドの意味やアルゴリズムに関する情報を含むように拡張するためには，多様な情報を表現できることが重要である．

表2: モジュールの持つ情報

クラスの識別子
そのクラスの識別子．そのクラスのオブジェクトが参照するオブジェクトのクラスの各識別子．そのクラスの上位クラスの識別子．
メソッドの形式
(つまり，メソッドの識別子，戻り値のクラス，各引数の識別子，各引数のクラスである．) そのクラスのオブジェクトが持つ各メソッドの形式．そのクラスのオブジェクトから呼び出される各メソッドの形式．
仮属性の形式
(つまり，各属性の識別子，クラスである．) そのクラスのオブジェクトが持つ各仮属性の形式．そのクラスのオブジェクトが参照するオブジェクトが持つ各仮属性の形式．
静的オブジェクトの形式
(つまり，各オブジェクトの識別子，クラスである．) そのクラスに関連する各静的オブジェクトの形式．そのクラスのオブジェクトが参照する各静的オブジェクトの形式．
実属性の形式
(つまり，各属性の識別子，クラスである．) そのクラスのオブジェクトが持つ各実属性の形式．
その他
そのクラスのオブジェクトのメソッド内で呼び出される各メソッド．そのクラスのオブジェクトのメソッド内で参照される実属性．そのクラスのオブジェクトのメソッド内で参照される各静的オブジェクト．そのクラスの上位クラス．

5.4 情報量の具体的な定義

現時点では，情報の要素の大小関係についてよくわかっていない．そこで，ここでは，最も単純な情報量の定義を与える．つまり，述語の1節で表される情報は，平等にその量が1であると考えることにする．従って，情報量 $q'(I_1)$ は，I_1 に含まれる節の個数になる．この定義は，3.2で述べた3条件をもちろん満たしている．

5.5 尺度の定義

以上の定義により，抽象的情報局所性の定義式(2)は，次のように具体化できる．

$$
\begin{aligned}
& h_{\langle I',q'\rangle} \\
&= \frac{\sum_{i\in E'} q'(i)}{\sum_{i\in E'}\left(q'(i)\sum_{m\in\{m|i\in I'(m)\}} q'(I'(m))\right)} \\
&= \frac{|E'|}{\sum_{i\in E'}\sum_{m\in\{m|i\in I'(m)\}}|I'(m)|} \qquad (3)
\end{aligned}
$$

表 3: 情報の形式的な表現

class(X).	Xがクラスの識別子である.
method(X, C).	Xが, Cと言うクラスのメソッドの識別子である.
parameter(X, M).	Xが, Mと言うメソッドの引数の識別子である.
vattribute(X, C).	Xが, Cと言うクラスのオブジェクトの仮属性である.
object(X).	Xが, 静的オブジェクトの識別子である.
attribute(X, C).	Xが, Cと言うクラスのオブジェクトの実属性である.
is(X, C).	Xと言う属性, または, Xと言うメソッドの戻り値, または, Xと言う引数, または, Xと言う静的オブジェクトが, Cと言うクラスである. または, Cと言うクラスがXと言うクラスの上位クラスである.
use(M, X).	Mと言うメソッド内で, Xと言う静的オブジェクト, または, Xと言う実属性が参照される. または, Xと言うメソッドが呼び出される,

ただし, E'は, ソフトウェアの持つ情報の中で, 変更の可能性のある情報の要素の集合である. つまり, 節の集合$\bigcup_m I'(m)$から, 言語に組み込まれているクラスや標準ライブラリに含まれるクラスに関する節を除いた部分集合である. $|I_1|$は, I_1に含まれる節の個数である. 以上が, 提案する情報局所性尺度の定義である. 値の算出の例は6章で示す.

6 尺度の適用例

小規模で同一の機能を持つ2個のソフトウェアA, Bを比較した例を述べる. それらは, ワークステーション上で使用する単純なアラームクロックのソフトウェアである. look と言うメッセージを受信した時に, 設定されている時刻に達していれば, 警報を鳴らす. 何らかの理由で警報が鳴らせない時は, 画面を点滅させる.

6.1 アラーム・クロック A

図1Aに示すように, A には, Aclock, Alarm, Speaker, Display, Timer と言う静的オブジェクトがある. それぞれ, aclock, alarm, speaker, display, timer と言うクラスに属する. Aclock は, メソッド look が呼び出されると, Timer のメソッド read を呼び出して, 現在時刻

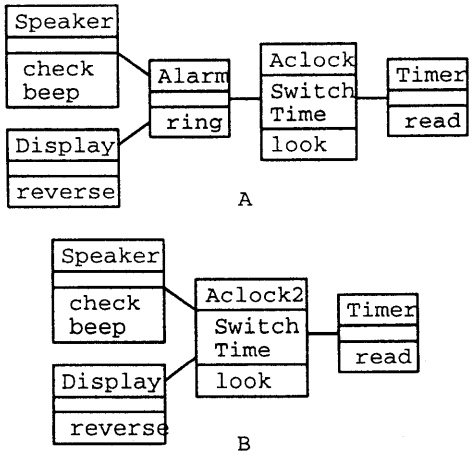

図 1: 比較するソフトウェアの例

を読み出す. そして, 設定時刻 Time に達していれば, Alarm のメソッド ring を呼び出す. Alarm は, メソッド ring が呼び出されると, Speaker のメソッド check を呼び出し, Speaker の状態を調べる. そして, Speaker が使用可能であれば, Speaker のメソッド beep を呼び出す. そうでなければ, Display のメソッド reverse を2回続けて呼び出し, 画面を点滅させる. 図には明示されていないが, A には, 上の5個のクラスの他に, time(時間), int(整数), bool(真偽値) と言う3個のクラスがある. 8個のクラスの中で, speaker, display, timer, time, int, bool は, 予め定義されていた標準的なクラスである. aclock と alarm だけが新たに定義された.

A について新たに作成された部分の情報局所性を計測してみる. aclock と alarm が持つ情報は, それぞれ, 表4と表5のようになり, 情報量は 36 と 25 となる. それらの中で, 変更の可能性のある情報の要素は, 6個の標準クラスに関する要素を除いた 29 個である. 各要素については, 関連するモジュールの規模の和が 36, 25, (36+25) のどれかになる. それを求めて計算すると, $h_{\langle I',q'\rangle} = 29/1056 \simeq .0275$ となる.

6.2 アラーム・クロック B

図 1B に示すように, B は A とよく似ている. A における Aclock と Alarm とが, 1個のオブジェクト

表 4: aclock の持つ情報

class(aclock). vattiribute(Switch, aclock). vattiribute(Time, aclock). method(look, aclock). object(Aclock). is(Switch, bool). is(Time, time). is(Aclock, aclock).
class(alarm). method(ring, alarm). object(Alarm). is(Alarm, alarm). class(timer). method(read, timer). object(Timer). is(read, time). is(Timer, timer). class(time). method(operator<, time). parameter(right, operator<). is(operator<, bool). is(right, time). class(bool). object(False). is(False, bool).
attribute(Switch0, aclock). attribute(Time0, aclock). is(Switch0, bool). is(Time0, time). use(look, Switch0). use(look, Time0). use(look, Timer). use(look, read). use(look, operator<). use(look, Alarm). use(look, ring).

表 5: alarm の持つ情報

class(alarm). method(ring, alarm). object(Alarm). is(Alarm, alarm).
class(speaker). method(check). method(beep). parameter(len, beep). object(Speaker). is(check, bool). is(len, int). is(Speaker, speaker). class(display). method(reverse). object(Display). is(Display, display). class(int). object(const100). is(const100, int). class(bool).
use(ring, Speaker). use(ring, check). use(ring, beep). use(ring, const100). use(ring, Display). use(ring, reverse).

Aclock2 に融合している点だけが異なっている.

経験的に言うと, B は A よりも変更が困難である. A においては, 例えば, timer を使用していることが aclock 内に隠蔽されていて, speaker や display を使用していることが alarm 内に隠蔽されている. 従って, timer, speaker, display のどれかを別のクラスに換えるとしても, その影響の波及する範囲は小さいと考えられる. 一方, B においては, 新たに作成された部分がモジュールに分割されていない, つまり, 情報隠蔽が行われていない. 変更の影響が波及する範囲が A よりも大きいと言える.

B についても A と同様に情報局所性を計測してみる. aclock2 の規模は, A における aclock と alarm の規模の和にほぼ等しく, 51 となる. 変更の可能性のある情報の要素も, A とほぼ等しく, 23 個である. 但し, A と異なり, どの要素についても, 関連するモジュールの規模の和が 51 となる. 結局, $h_{\langle I',q'\rangle} = 23/1173 \simeq .0196$ となり, 期待通り A より小さくなる. B の方が変更が困難であると言う経験的な評価とよく合致する.

7 おわりに

オブジェクト指向ソフトウエアの変更容易性を評価するための尺度である情報局所性尺度を提案した. 提案した尺度を小規模なソフトウェアに適用したところ, 尺度の値と, 経験者の直観的な評価とがよく合致することがわかった.

今後は, より多くの例に対する尺度の適用, 情報の 1 つであるメソッドの意味の形式化, 情報量の定義についての再検討, 尺度の値の算出の自動化などについて研究を進める予定である.

参考文献

[1] Basili, V. R., Rombach, H. D.: "The TAME Project: Towards Improvement-Oriented Software Environments", IEEE Trans. on Soft. Eng., vol. SE-14, no. 6, pp. 758–772 (June 1988).

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

[3] Chidamber, S. R. and Kemerer, C. F.: "A Metrics Suite for Object Oriented Design", IEEE Trans. on Soft. Eng., vol. 20, no. 6, pp. 476–493 (June 1994).

[4] Coad, P. and Yourdon, E.: "Object-Oriented Analysis", Yourdon Press Computing Series, Prentice Hall (1990).

[5] Fenton, N. E.: "Software Metrics: A Rigorous Approach", Chapman and Hall (1991).

[6] Fenton, N. E.: "Software Measurement: A Necessary Scientific Basis", IEEE Trans. on Soft. Eng., vol. 20, no. 3, pp. 199–206 (March 1994).

[7] 羽生田 栄一 監訳: "オブジェクト指向方法論 OMT: モデル化と設計", トッパン (1992).

[8] Meyer, B.: "Object-Oriented Software Construction", Prentice Hall (1988).