特別研究報告

題目

複雑度メトリクスを用いたJavaプログラム品質特性の実験的評価

指導教官
井上 克郎 教授

報告者
岡井 洋樹

平成 13 年 2 月 22 日

大阪大学 基礎工学部 情報科学科

平成 12 年度 特別研究報告

複雑度メトリクスを用いた Java プログラム品質特性の実験的評価

岡井 洋樹

内容梗概

オブジェクト指向方法論に基づいて開発されたソフトウェアの品質を評価するために, 多くのメトリクスが提案されている. 現在提案されているメトリクスの多くは静的な複雑さ, つまり設計仕様書やソースコードのみから計測を行なうものであった. しかし, 動的なメトリクス , つまり実行されている状態のソフトウェアのオブジェクトの振舞いを評価するメトリクスの需要が高まってきている . Yacoub らによって , オブジェクト指向ソフトウェアに対する動的な複雑度メトリクスが提案されたが, その有効性の評価は不十分であった.

本研究ではまず, 動的メトリクスである OQFS(Object Request for Service) と OPFS(Object Response for Service) と, 静的メトリクスである MM(Method-Method Interaction) と ICP(Information Flow-Based Coupling) の比較を行なった. 実験の結果, 静的メトリクスより動的メトリクスの方がソフトウェアの複雑さを評価する際に有効である事が確認できた. 次に動的メトリクスの値がクラスの再利用性を予測できるかどうかを実験的に評価を行なった. 実験の結果, 動的メトリクスによりクラスの再利用性の評価が行なえる事が確認できた.

主な用語

オブジェクト指向開発
複雑度メトリクス
JAVA
実験的評価

目 次

1 まえがき **4**

2 準備 **6**

2.1 メトリクス 6

2.2 オブジェクト指向開発 7

2.3 オブジェクト指向メトリクス 8

2.4 Chidamber らのメトリクス 10

2.5 ロジスティック回帰 10

2.6 再利用性 11

3 複雑度メトリクス **12**

3.1 動的メトリクス 12

3.2 Yacoub らの動的メトリクス 12

3.2.1 用語と定義 12

3.2.2 $EOC_x(o_i, o_j)$ と $OQFS$ 13

3.2.3 $IOCx(o_i, o_j)$ と $OPFS$ 14

3.2.4 シナリオプロファイルとの統合 16

3.2.5 具体例 16

3.3 Yacoub らによる動的メトリクスの評価 17

3.4 静的メトリクス 18

3.4.1 MM 18

3.4.2 ICP 18

4 実験 **19**

4.1 実験背景 19

4.2 実験対象 19

4.3 開発状況 22

4.4 シナリオ 22

5 分析 **24**

5.1 動的メトリクスと静的メトリクス 24

5.2 動的メトリクスと静的メトリクスの相関 27

5.3 再利用性 27

6　まとめ **30**

謝辞 **31**

参考文献 **32**

1　まえがき

近年, オブジェクト指向方法論に基づいてソフトウェア開発が盛んに行なわれている. 従来の手続き型のソフトウェア開発に際にもみられるように, オブジェクト指向方法論に基づくソフトウェア開発の際に, 各開発工程において適切なメトリクスを用いてソフトウェアの品質や開発管理を行なう事は, ソフトウェアの開発管理を効果的に行なう上で重要になってきている. またソフトウェア開発の初期の開発工程においてソフトウェアの品質の評価を行なう事により, のちの段階の開発工数の見積りが容易になる事や予算の削減などに繋がる事が云われている. しかしオブジェクト指向方法論に基づいて開発されたソフトウェアに対して, 従来の手続き型で開発されたソフトウェアを測るメトリクスでは不十分なものがある. そこで最近オブジェクト指向方法論に基づいて開発されたソフトェアに対するメトリクスが提案されてきている.

オブジェクト指向ソフトウェアに関するメトリクスとして,Briand らにより提案された MM と Lee らにより提案された ICP がある. これらのメトリクスは静的な複雑度メトリクス, つまり設計書やソースコードから情報を得て計測を行なうメトリクスである. 2 つのメトリクスに共通する事は, どちらもクラスのメソッド呼びだしを基準に測っている事である. 一方, 静的な複雑度メトリクスの他に最近では動的な複雑度メトリクスも提案されている.

Yacoub らによってオブジェクト指向方法論に基づいて開発されるソフトウェアに対して動的な複雑度メトリクスが提案された. この動的な複雑度メトリクスは, ある入力データの実行時においてクラス間のメッセージの送受信回数や状態遷移の複雑度を評価するものである. しかし , この動的複雑度メトリクスは提案のみがされており本来評価すべき, 動的な複雑度と静的な複雑度の違いの評価や, 保守性や理解容易性や再利用性やエラー伝播と発生可能性という品質評価において議論されておらず, 不十分なものとなっている.

本研究では, 先程述べた動的な複雑度と静的な複雑度の違いの評価や, 保守性や理解容易性や再利用性やエラー伝播と発生可能性という品質評価においてソフトウェア工学実験を計画し, 教育環境におけるオブジェクト指向開発プログラム開発プロセスから収集されたデータを用いる事により,Yacoub らの動的複雑度メトリクスの評価を行なう事にする. 具体的には,Yacoub らにより提案されている OQFS,OPFS の 2 つを今回使用する動的複雑度メトリクスと定め, 静的メトリクスとしては Briand らにより提案された MM と Lee らにより提案された ICP の 2 つのメトリクスを静的メトリクスと定め, それぞれのメトリクスの計測を行なう. 計測対象としては, ある企業の教育環境におけるオブジェクト指向プログラム開発プロセスを用いて,OQFS,OPFS,MM,ICP のそれぞれを計測し, 動的な複雑度メトリクスと静的な複雑度メトリクスの比較評価を行ない結果の考察を行なう. 次に OQFS を使用し品質特性のひとつである再利用性についても評価し結果の考察を行なう.

結果として, 動的な複雑度メトリクスの方が静的な複雑度メトリクスよりソフトウェアの品質を測る上では有効である事が確認された. また,OQFS と OPFS の相関関係から OQFS の値が高いクラスは再利用性が低い事が確認された.

以降 2 章では準備として基本的な用語の説明を行なう. 次に 3 章では複雑度メトリクスの説明を行なう. そして 4 章では今回使用した実験対象の説明を行なう.5 章では実験後得られたデータの分析を行ない, 最後に 6 章でまとめと今後の課題について述べる.

2 準備

2.1 メトリクス

ソフトウェアメトリクス (software metrics) は, ソフトウェアのさまざまな特性を判別する客観的な定量的尺度として広く知られ, ソフトウェアがシステムの動作環境において発揮するソフトウェア品質特性の定量的尺度を与えている. ソフトウェアメトリクスは次のように定義できる [18].

> 「ソフトウェアメトリクスはソフトウェアプロダクトおよびソフトウェア開発・保守のプロセスに関係する定量的尺度である.」

リソース割り当てやコスト見積り, レビューチェック項目やテスト網羅度の目安となるメトリクスとして, 量のメトリクスがある. 量のメトリクスはさまざまなメトリクスの基礎となる. 代表的な例を表 1 に示す.

表 1: 量のメトリクス

メトリクス	概要	対象
LOC(Lines Of Codes)	ソースコードの行数	アプリケーション, クラス, メソド, ライブラリ
Function Point	外部仕様からみた機能量	アプリケーション

LOC は, プログラム記述スタイルやプログラム言語に依存するといった欠点もあるが, 測定が簡単であり, 「目に見える量である」という理由で幅広く使われている. また, LOC をより厳密に定義したものでセミコロンの数も, メトリクスとして提案されている.

Function Point は, 外部仕様から見た機能を抽出し, 各々の機能の複雑さを評価し, さらにシステムの特性を評価して加味するという, いくつかのステップによって行なわれる. 設計段階での計測が可能であり, 工数見積りに精度良く用いることができる.

量のメトリクスに対し, 複雑度メトリクスは分析・設計・コーディングの指針, あるいはプロダクトの作り直し箇所の判断に用いることができる. 代表的な例を表 2 に示す.

McCabe のメトリクス, Halstead のメトリクスどちらも, 手続きの手順の複雑度を表すものである [8].

表 2: 複雑度メトリクス

メトリクス	概要	対象
McCabe のサイクロマチック数	制御フローグラフにおける基本パス数	手続き
Halstead のメトリクス	オペレータとオペランドの種類数と出現数に基づいて計算	手続き

2.2 オブジェクト指向開発

ソフトウェアの再利用に適した開発方法としてオブジェクト指向開発がある. 従来の (非オブジェクト指向での) 開発でも, 再利用のためにソフトウェア部品を整理・分類した「ライブラリ」が使用されていた. ライブラリを用いた場合, プログラム全体の処理の流れなどの主要な部分は開発し, ライブラリから必要な部品をもってきて組み合わせる, という形態になる. オブジェクト指向開発ではライブラリの利用は大きな課題であった. 莫大な数に達するライブラリの中から, 実際に再利用できるクラスを限定することが困難であったためである. しかし, オブジェクト指向開発において,「フレームワーク」という, 特殊なライブラリを用いることによって, その困難を解決した. フレームワークは, 主要な基本機能を提供するクラスを集めたライブラリである. フレームワークを用いる場合, 開発者はまず, フレームワークから開発しようとしているプログラムの主要部分となるクラスを捜し出す. 次に, そのクラスに対して, 新規に開発したクラスを組み合わせる. 主要な部分を再利用するために, 従来の開発よりも大きな割合で再利用が行なえるようになった. 特に GUI(Graphical User Interface) フレームワークは, MFC(Microsoft Foundation Class) のように商品化がなされており, 実用化が最も進んでいる.

オブジェクト指向開発に従ったプログラム開発プロセスは大きく分けると次のようなフェーズからなる.

1. 要求仕様定義
   システムの要求仕様を定義する.

2. 設計
   システムをオブジェクトおよびそれらの関係として設計する.

3. 実装
   システムを実装する. すなわち, プログラムをコーディングする.

設計に際して, 分析手法として, OMT 法 [14], Booch 法 [5], デザインパターンなどが用いられている.

実装には, オブジェクトという概念を素直に表現できて, オブジェクト指向の利点を最大限引き出すことができる言語, すなわちオブジェクト指向言語を用いるべきである. 代表的なものに, Smalltalk, C + + , Java などがある.

オブジェクト指向開発プロセスを管理するために, さまざまなモデルが提唱されている. 従来のウォーターフォールモデルも用いられるし, スパイラルモデルや, ラッピドプロトタイピングなど, 新しいモデルも登場している. しかし, 現在, オブジェクト指向開発プロセスを管理するための方法論が確立されていない. また, 従来の機能分割の方法論と比較して有効性の評価が十分に行なわれているとは言えない.

2.3 オブジェクト指向メトリクス

オブジェクト指向で開発されたソフトウェアに対して, 2 . 1 で示したような従来のメトリクスでは, 不十分である. そのため, オブジェクト指向特有のメトリクスが提案されてきている. 量のメトリクスの代表例を表 3 に示す. これらのメトリクスは, 分析, 設計段階においても用いることができ, 計測も簡単で直観的なメトリクスであるため, 従来の LOC などと併用して使われていくであろう.

表 3: オブジェクト指向特有の量のメトリクス

メトリクス	概要	対象
クラス数	クラスの全定義数	アプリケーション, ライブラリ
サブシステム数	サブシステムの数	アプリケーション
メソド数	あるクラスメソドの数	クラス
プロパティ数	あるクラスの属性数 + メソド数	クラス

また, 従来の複雑度メトリクスである McCabe のメトリクスや, Halstead のメトリクスは, 一つの手続き (メソド) の内部の複雑さの指標となりうるが, オブジェクト指向ソフトウェア全体の複雑さを表すのには不十分である. そこで, Chidamber らによって 6 つのメトリクスがオブジェクト指向特有の複雑度メトリクスとして提案されている. これらの定義を示す. なお, 対象はすべてクラスであり, その値が大きいほど, 複雑であることになる.

WMC(Weighted Method per Class) ; クラスあたりの重み付きメソド数

: クラスのメソドがどれも同じくらいの複雑さであると仮定できれば, クラスのメソッドの数.

DIT(Depth of Inheritance Tree of a class) ; 継承木の深さ

: クラスの派生関係が木であると仮定できれば, そのクラスの木の中での深さ.

NOC(Number Of Children) ; 子クラスの数

: そのクラスから直接派生しているクラスの数.

CBO(Coupling Between Object class) ; クラス間の結合

: そのクラスが「結合」しているクラスの数. あるクラスが他のクラスに結合しているとは, あるクラスが他のクラスの属性やメソドを用いていることをいう.

RFC(Reference For a Class) ; クラスに対する反応

: あるクラスのメソドの集合と, 各メソドで呼び出す他のクラスのメソドの集合の和集合の要素数.

LCOM(Lack of Cohesion in Methods) ; メソドの凝集の欠如

: あるクラスのメソドのすべての組み合わせのうち, 参照する属性に共通するもののない組み合わせの数から, 共通するものがある組み合わせの数を引いたもの. ただし, 0 より小さいときは 0 とする.

それぞれのメトリクスに対して, その特徴を述べる.

- WMC : クラスの開発や保守に必要な量の指標となる.
- DIT : 深いと再利用性が向上する一方, 理解用意性や保守性が下がる.
- NOC : クラスがその設計に対して持つ潜在的な影響を示唆する.
- CBO : 保守性・拡張性に影響する.
- RFC : テスト, デバッグの複雑さに関係がある.
- LCOM : メソド間におけるインスタンス変数の非共有の値. すなわち凝集度合の欠如を示す.

2.4 Chidamber らのメトリクス

Basili らが論文 [16] のなかで, Chidamber らのメトリクスとプログラムの品質との関係を実験的に評価している. その中では, 6 つうち LCOM を除く 5 つのメトリクスは, プログラムの品質を予測するのに従来の複雑度メトリクスより役立つことが示されている (表 4) .

表 4: Chidamber らのメトリクスと従来のメトリクスの比較

	OO metrics	Code metrics
Completeness	93%	86%
Correctness	92%	86%

OO metrics : Chidamber らのメトリクス

Code metrics : 従来のメトリクス

Completeness : 全フォールトの中で, メトリクスによってフォールトがあると予測されたクラスのエラー数の割合

Correctness : フォールトがあると予測された中で, 正しかったエラーの割合.

ただし, この Basili らの実験で, プログラムの品質はエラーの個数であった. しかし, 同じエラーであっても, その難易度には差があると予測でき, プログラムの複雑度を単にエラー数で評価するのには問題があると考えられる.

また, 再利用されたクラスと新規開発のクラスを同等に扱っているため, 大規模な再利用を行なう開発での適用方法に関して検討が必要である.

2.5 ロジスティック回帰

文献 [12] では, ロジスティック回帰分析がプログラムのフォールト発生を予測するのに用いられている. 本研究でもこの手法を用いる. メトリクスの fault-prone 予測性能 (対象のクラスにフォールトが発生するかを予測する性能) を評価するためには, まず, クラスのメトリクス値を入力とし, 真偽値 (予測) を出力する関数を作る必要がある. 出力の真偽値はクラスがフォールトを持つか持たないかを示す. 関数に観測されたメトリクス値を与えて予測を行なう. その後, 予測モデルと実際に観測されたフォールトを比較することで, モデルの正確さを判定し, メトリクスの性能を評価する.

単変量ロジスティック回帰モデルは以下の関係式を用いる.

$$P(X_1) = \frac{1}{1 + \exp(-(C_0 + C_1 \cdot X_1))}$$

ここで, P は与えられたクラスにエラーが見つかる確率であり, X_1 はクラスのメトリクスの値である. もし与えられたメトリクス値が P を 0.5 以上にするなら, クラスはフォールトを持つ (fault-prone である) と予測する. この式において, $Z = C_0 + C_1 \cdot X_1$ とおけば, $P(Z) = 1/(1 + \exp(-Z))$ となる. この P と Z の関係は S 字カーブになる. このような S 字カーブは X_1 と P の関係が単調である限り, 2 値の分類に適用可能である.

係数 C_1 を決定する際には, 最尤度基準が用いられる. すなわち, 係数は観測された結果をもっともよく反映するような値が選ばれる.

2.6 再利用性

過去に，あるオブジェクト指向プラグラミング言語を用いてプログラムを作成した時に，そのプログラムが再利用が可能かどうかであるかの判断は非常に難しい．

しかし，再利用可能かどうかが判断できれば，コーディングの際の負担の軽減は測りしれないものがある．

ここでは，動的メトリクスの計測結果から再利用可能かどうかの判断が可能かどうかの実験的評価を行なう事にする．

3　複雑度メトリクス

3.1　動的メトリクス

これまでに様々なメトリクスがオブジェクト指向ソフトウェアの品質を評価するために提案されてきた．一般に，メトリクスは静的なものと動的なものの 2 種類に分類される．直観的には，静的なメトリクスとは，設計書やソースコードから収集される情報のみで計測し，動的なメトリクスとは，ソフトウェアの実行時の振舞いから収集される情報で計測されるメトリクスと定義される．

簡単な例を用いて，静的なメトリクスと動的なメトリクスの違いを説明する [6]．

(CASE1) クラス c_1 がクラス c_2 のあるメソッドを実行時に 10 回呼びだす．

(CASE2) クラス c_1 が異なる 10 種類のクラスの 10 種類のメソッドを実行時にそれぞれ 1 回ずつ呼びだす．

(CASE1) は「メソッドの呼び出し数」を計測するのに対し，(CASE2) は「呼び出されるメソッドの数」を計測している．Briand[1] らは，(CASE2) を今日使用されている一般的なメトリクスとして位置づけている．Yacoub らは (CASE1) を動的なメトリクスとして区別している．

3.2　Yacoub らの動的メトリクス

ここでは，Yacoub らが定義した動的複雑度メトリクスについて紹介する．このメトリクスは特定の入力データシナリオに対して以下の 2 点について計測し, 評価を行う.

- クラス間メッセージの送受信回数
- 状態遷移の複雑さ

本研究では, 前者のクラス間メッセージの送受信回数についての評価を行うためそのメトリクスに関して説明を行う. まず，メトリクスを定義する上で使用される用語とメトリクスの定義を述べる．更に，文献 [17] で述べられている，そのメトリクスがどのような設計品質の要素を評価するかをまとめる．

3.2.1　用語と定義

先ず，o_i はオブジェクト (すなわち何らかのクラスのインスタンス) を，O はシナリオの実行の間に関わったオブジェクトの集合を，$|Z|$ は集合 Z の要素数を，$\{\}$ はある集合の要素を，それぞれ表すものとする．

次に，シナリオ，シナリオの確率，シナリオプロファイル，メッセージ集合，シナリオ内の総メッセージを以下のように定義している．

シナリオ “x”: 入力データ，イベントによって実行されるオブジェクト間の一連の相互作用．

シナリオの確率 “PS_x”: すべてのシナリオに対するあるシナリオの実行頻度．

シナリオプロファイル: シナリオが実行される確率の集合．

メッセージ集合 “$M_x(o_i, o_j)$”: シナリオ x の実行の間，オブジェクト o_i からオブジェクト o_j に送られるメッセージ集合．

シナリオ内の総メッセージ “MT_x”: シナリオ x の実行の間，オブジェクト間で交換されるメッセージの総数．

3.2.2 $EOC_x(o_i, o_j)$ と $OQFS$

シナリオ x の実行中に交換されるメッセージの総数のうち o_i から o_j に送信されるメッセージの割合 $(\%)$ を Export Object Coupling$(EOC_x(o_i, o_j))$ と呼び，次式で定義される．

$$EOC_x(o_i, o_j) = \frac{|\{M_x(o_i, o_j)|o_i, o_j \in O \wedge o_i \neq o_j\}|}{MT_x} \times 100$$

EOC は 2 つの特定のオブジェクト間の結合関係を計測する．オブジェクト間の結合を評価することで，エラーを発生させている箇所の特定や強く結びついているオブジェクトの特定が可能となり，特にオブジェクト間の相互作用のテストに有用である．

更に，EOC を用いて Object Request for Service$(OQFS)$ を定義している．OQFS はシナリオ x の実行中に交換されるメッセージの総数のうち o_i から他の全てのオブジェクトに送信されるメッセージの割合 $(\%)$ を表し，次式で定義される．

$$\begin{aligned} OQFS_x(o_i) &= \frac{\left|\left\{\bigcup_{j=1}^{|K|} M_x(o_i, o_j)|o_i, o_j \in O \wedge o_i \neq o_j\right\}\right|}{MT_x} \times 100 \\ &= \sum_{j=1}^{|K|} EOC_x(o_i, o_j) \end{aligned}$$

ここで，K はオブジェクトの集合である．

3.2.3 $IOCx(o_i, o_j)$ と $OPFS$

シナリオ x の実行中に交換されるメッセージの総数のうちオブジェクト o_j が送信したメッセージの中で o_i が受信したメッセージの割合 (%) を Import Object Coupling($IOCx(o_i, o_j)$) と呼び，次式で定義される．

$$IOC_x(o_i, o_j) = \frac{|\{M_x(o_j, o_i)|o_i, o_j \in O \wedge o_i \neq o_j\}|}{MT_x} \times 100$$

EOC と同様に IOC は 2 つの特定のオブジェクト間の結合関係を計測する．

更に，IOC を用いて Object Response for Service(OPFS) を定義している．OPFS はシナリオ x の実行中に交換されるメッセージの総数のうち他の全てのオブジェクトから o_i に送信されるメッセージの割合 (%) を表し，次式で定義される．

$$\begin{aligned} OPFS_x(o_i) &= \frac{\left|\left\{\bigcup_{j=1}^{|K|} M_x(o_i, o_j)|o_i, o_j \in O \wedge o_i \neq o_j\right\}\right|}{MT_x} \times 100 \\ &= \sum_{j=1}^{|K|} EOC_x(o_i, o_j) \end{aligned}$$

ここで，K はオブジェクトの集合である．

Yacoub らは EOC, IOC の値は以下のような設計品質の要素を評価すると述べている．

Maintainability(保守性)：ある特定のオブジェクトとの EOC, IOC が高いオブジェクトが属するクラスは，保守による変更に注意を要する．また，そのクラスの変更に伴い，結合しているオブジェクトにも変更が必要となる．

Understandability(理解容易性)：特定のオブジェクトから非常に多くのメッセージを受信しているオブジェクトは，動的な振る舞いがその特定のオブジェクトに強く依存しているため，そのオブジェクトのみを単独に理解することは困難である．

Reusability(再利用性)：特定のオブジェクトへの EOC, IOC が高いオブジェクトは，別のオブジェクトに強く依存し，頻繁に他のオブジェクトのサービスを要求しているため，再利用がほぼ不可能であると考えられる．EOC, IOC が高いオブジェクトの組は，2 つまとめて使用される．

Error Propagation/Proneness(エラーの伝播と発生可能性)：EOC が高いオブジェクトは，特定のオブジェクトに頻繁にメッセージを送信するため，送信元のクラスにエラーが含まれているとそのエラーを含むメッセージを送信先のオブジェクトへ伝播してしま

う．また，IOC が高いオブジェクトは，そのオブジェクト自体がアプリケーションの故障の原因となる．これらのオブジェクトのサービスが他のオブジェクトに頻繁に必要とされているからである.

3.2.4 シナリオプロファイルとの統合

上述したメトリクスは，特定の実行シナリオに対して定義され，シナリオの実行確率を導入することでメトリクスを拡張することが可能であるとされている．IOC, $OPFS$, EOC, $OQFS$ を異なるシナリオに対して計測し，それぞれのシナリオ確率を定義することによって求められる．拡張されたメトリクスの定義は以下のようになる．

$$IOC(o_i, o_j) = \sum_{x=1}^{|X|} PS_x \times IOC_x(o_i, o_j)$$

$$OPFS(o_i) = \sum_{x=1}^{|X|} PS_x \times OPFS_x(o_i)$$

$$EOC(o_i, o_j) = \sum_{x=1}^{|X|} PS_x \times IOC_x(o_i, o_j)$$

$$OQFS(o_i) = \sum_{x=1}^{|X|} PS_x \times OQFS_x(o_i)$$

3.2.5 具体例

表 5: シナリオの例

x	M_x	MT_x
$x(T_1)$	$O_1 \rightarrow O_2 \rightarrow O_3$	2
$x(T_2)$	$O_2 \rightarrow O_3 \rightarrow O_1$	2
$x(T_3)$	$O_1 \rightarrow O_2$	1

ここでは, 先に定義した用語と動的メトリクスの計測手法について具体的な例をあてはめて説明する.

表 5 において,x はシナリオを,T_1 はテストデータを,M_x はメッセージ集合を,MT_x はシナリオ内の総メッセージを表す. つまり, 例えば,$x(T_1)$ はテストデータ T_1 を与えられたシナリオを表す. $x(T_2)$,$x(T_3)$ もそれぞれテストデータ T_2,T_3 に対するシナリオとなる. 表 5 の例ではシナリオ $x(T_1)$ を与えた時に実際の動作がオブジェクト O_1 から O_2,O_2 から O_3 にメッセージが送信されたことを表す. このときのシナリオ内の総メッセージ数は 2 となる. 他のシナリオを与えた時も同様にシナリオ内の総メッセージ数を計測することができる.

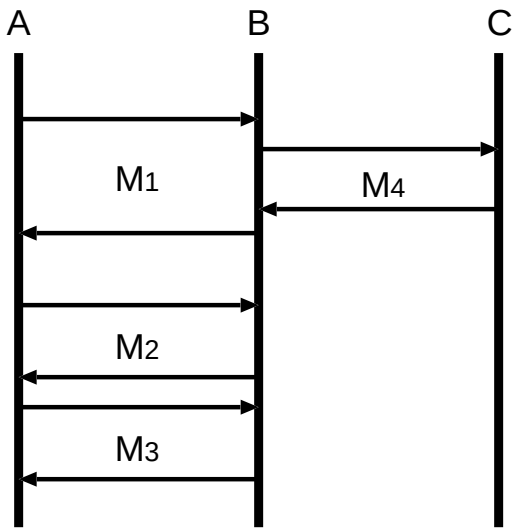

図 1: 動的メトリクスの計測例

表 6: Yacoub らの実験結果

	RS	CD	CG	AR	VT
OPFS	0.08	0.5	0.25	42.35	56.82
OQFS	0.27	0.27	0.46	53.53	45.47
CBO	13.5	23.1	28.8	17.3	17.3

次に, 図 1 を使用して, 動的メトリクスの計測方法について説明する. この例で A,B,C はそれぞれオブジェクトを表し,M_1,M_2,M_3 はそれぞれ送信されるメッセージを表す. 例えば M_1 は, オブジェクト A からオブジェクト B に送信されたメッセージを表す. この例では, シナリオ内の総メッセージは 4 となり, オブジェクト A については,A から B にメッセージを 3 回送信しているので,

$$OQFS = \frac{3}{4} \times 100 = 75(\%)$$

となる. オブジェクト B,C についても同様に計測し,25,0 となる.OPFS はこの逆となるので, オブジェクト A,B,C について 0,75,25 となる.

3.3 Yacoub らによる動的メトリクスの評価

Yacoub らが行った実験の結果を表 6 に示す．彼らはペースメーカーシステムの 5 つのコンポーネント (RS, CD, CG, AR, VT と名付けられているいる), に対して動的，静的それぞれのメトリクスを適用した．彼らは，この結果から，動的メトリクスは AR, VT の値が高

くなっているのに対し，静的メトリクスでは，CD, CG の値が高いが他の値と比べてあまり特徴的でないと評価している．

但し，これらの値はメトリクスの計測値に過ぎず，メトリクス間の相関関係や実際の開発におけるエラー数との関係の評価は行っていない．

3.4 静的メトリクス

静的メトリクスとは, 設計書やソースコードから収集される情報で計測するメトリクスの事である. 以下では数ある静的メトリクスのうち, 今回動的メトリクスの比較対象として選んだ MM と ICP について説明を行なう.

3.4.1 MM

MM とは Method-Method interraction の略であり,2 つのクラス間でのメソッド呼びだしの回数を計測し, それぞれのメソッド呼び出しの回数をクラスごとに合計した値である. 計測の際に, 引数の数や種類の数の計測といったものは定義に含まれておらず, 計測対象のクラスのソースコード中に, 違う他のクラスのメソッド呼びだしが行なわれた時に計測を行なう．

3.4.2 ICP

ICP とは Information Flow-Based Coupling の略であり,2 つのクラス間でのメソッド呼びだしの回数を計測し, そのメソッドの引数の重みを考慮して出す値である. こちらの場合は MM とは異なり, 引数の重みも考慮に入れて計測を行なう.

4 実験

4.1 実験背景

まず実験対象から得られる動的メトリクスの値と静的メトリクスの値を比較して分析を行う.

次に,Yacoub らは動的メトリクスは以下のような設計品質特性を評価すると述べている.

1. Maintainability(保守性)
2. Understandability(理解容易性)
3. Reusability(再利用性)
4. Error Propagation/Proness(エラーの伝播と発生可能性)

本論文では 3 の再利用性についても実験を行うものとする.

4.2 実験対象

実験対象として選んだソースコードは, 日本ユニシス株式会社で 2000 年 7 月に行われた新人研修の際に作成されたオークションシステムのプログラムである. その新人研修では, 研修者は 36 チームに別れ,1 チームは 4 名から 5 名で構成されている. そして, それぞれのチームで独立してオークションシステムの開発を行っている. オークションシステムの詳細な仕様に関しては, あらかじめ定義されており, その定義に従ってそれぞれのチームが開発を行った. 開発されたオークションシステムのソースコードの使用言語は JAVA であり, 処理系は JDK1.2.2 である. オークションシステムの構成として,1 つのオークションシステムはそれぞれ 29 種類のクラスから作成されており, それぞれのクラスは行数で表すと 10 行程度のものから 500 行程度のものまでさまざまである. その行数の中には, 空白やコメント等も含まれている.

ここで, それぞれのコンポーネントはどのような機能を実装していうかを表 7 に示す.

表 7: 作成するコンポーネント

コンポーネント	内容
AddCategoryDialog	カテゴリ追加ダイアログ
AuctionClient	オークションシステムのクライアント
AuctionManager	オークションマネージャリモートオブジェクトのインターフェース
AuctionManagerImpl	オークションマネージャの実装
AuctionServer	オークションサーバ
Bid	入札リモートオブジェクトのインターフェースを定める
BidDialog	入札ダイアログ
BidImpl	入札オブジェクトの実装
Category	カテゴリリモートオブジェクトのインターフェースを定義する
CategoryImpl	カテゴリオブジェクトの実装
CloseBidDialog	入札締切ダイアログ
DuplicateCategoryException	カテゴリ名重複例外
DuplicateUserIDException	会員 ID 重複例外
Exhibit	出品物リモートオブジェクトのインターフェースを定める
ExhibitDialog	出品ダイアログ
ExhibitDisplayDialog	出品物詳細表示ダイアログ
ExhibitImpl	出品物オブジェクトの実装
InappropriateBidException	不適切入札例外
InformationDialog	情報表示ダイアログ
InvalidCategoryException	不正カテゴリ例外
InvalidCredentialException	認証例外
InvalidExhibitException	不正出品物例外
InvalidMemberException	不正会員例外
Member	会員リモートオブジェクトのインターフェースを定める
MemberCredential	会員の認証情報リモートオブジェクトのインターフェースを定義する
MemberCredentialImpl	会員の認証情報リモートオブジェクトの実装
MemberImpl	会員オブジェクトの実装
MemberRegistrationDialog	会員登録ダイアログ
Node	カテゴリに追加できるものを表現する基底クラス

前述しているオークションシステムは次のような工程を踏み作成されている.

- 仕様の理解
- 仕様レビュー
- 設計
- 設計レビュー
- テストケース作成
- テストケースレビュー
- コーディング
- コードレビュー
- テスト
- デバグ

そして, 作成されたオークションシステムは次に挙げる機能が実装されている.

1. 会員登録
   個人情報をユーザーが入力する機能である. 個人情報としては, 氏名, 電子メールアドレス, 電話番号,ID, パスワードがある. この機能を使用する事により, オークションシステムに対して出品, 入札あるいはカテゴリ追加といった作業を行う事ができるようになる.

2. 出品
   会員に入札してもらう為に, 出品物の情報を入力する機能である. 出品物の情報としては,ID, パスワード, 出品物名, 出品物の説明, 入札の開始価格, 支払い方法, 配送方法がある.

3. 入札
   出品されている出品物を見て, 入札の情報を入力する機能である. 入札の際の情報としては,ID, パスワード, 入札価格がある.

4. 入札締め切り
   出品者が入札した自分の出品物の状況を見て, 入札を締め切る事である. 入札締め切りの際の情報としては,ID, パスワードがある.

5. カテゴリ追加

出品物のカテゴリをオークションシステムに追加する機能である. カテゴリ追加の情報としては,ID, パスワード, カテゴリ名がある.

上述されている事に加えて, 入力ではないが追加されたカテゴリの中に入っている出品物を確認できる機能や, カテゴリを選択中に選択しているカテゴリから抜け出る機能がある. これら 2 つの機能については, 会員情報は必要ないので会員登録していなくても自由に選択する事が可能である.

4.3 開発状況

オークションシステムの開発状況について説明をする.

ここで開発されたオークションシステムは, チームに分かれた個人それぞれに割り当てられたパーソナルコンピュータ上で開発が行われた. そのパーソナルコンピュータやサーバーは, 同一のネットワーク上に接続されており, サーバーは AM10:00 から PM20:00 の間,1 時間に一度個人の開発しているパーソナルコンピュータのバックアップを取る事により, 自動的に個人それぞれのソースコードを収集する. 今回実験対象として使用するソースコードは, サーバーが収集した大量のソースコードのうち, チームとして最終にできあがったソースコードを実験対象にする. 実験対象とするソースコードは, チームが 36 チーム存在するので 36 種類のオークションシステムが存在するわけだが, それは開発期限が決められている事もあり全てが全て完全に動作できる状態ではない. そこで今回使用するソースコードは, 実際に動作させてみて問題無く動作するものに限定した. 結局計測対象となったソースコードは 7 チーム分となった. 決定した 7 チームを使用し, 動的メトリクスと静的メトリクスを計測し分析していくことにする. しかし動的メトリクスを計測するには, 実際に実行している状況からログを取って計測する事になるので, 入力データであるシナリオが必要になる. 以下でシナリオの説明を行う.

4.4 シナリオ

ここでは, 動的メトリクスを計測する為に必要なシナリオについて説明する. まず, ここでいうシナリオについて定義しておく. シナリオとは, オークションシステムが実装している機能を使用し, それらを組み合わせて 1 つの仮想的なオークションの流れを表すものである. ここでいうオークションシステムが実装している機能とは, 会員登録, 出品, 入札, 入札締め切り, カテゴリ追加, カテゴリの中を見る, カテゴリを出るという 7 つの機能である. 簡単なシナリオの例を表すと,

会員登録→出品→入札→カテゴリ追加→...

という様な機能の繰り返し作業を 1 つのシナリオとして定義する. ここで, シナリオにユーザとして 10 人を定義し, それぞれが会員登録したり出品したりといった作業を繰り返すものとする. 今回動的メトリクスを計測する為にシナリオを 30 種類用意した. この 30 種類のシナリオを用いる事により, オークションシステムのクラスを全て使用するように作成した. 具体的には, 仕様書やソースコードを使用する事により, それぞれのクラスがどの機能を使用した時に実行されるのかを確認した後シナリオの作成をしている. そしてそれぞれのシナリオは, 会員登録をして終わるだけのシナリオから, エラー処理を含むものまで様々なものがある. そのような様々なシナリオを使用し, 動的メトリクスを計測する.

5 分析

実験により収集されたデータを使用し, ここでは分析を行っていく. 分析は, 大きくわけて 2 種類ある.

1. 動的メトリクスと静的メトリクスの比較

2. 再利用性の評価

1 は, 動的メトリクスとして OQFS と OPFS, 静的メトリクス MM と ICP を計測し, それらの分析を行う.

2 は,1 で求めた動的メトリクス OQFS を利用し, クラスを入れ替え実験結果から分析を行う. 以下では, それぞれについて分析を行っていく事にする.

5.1 動的メトリクスと静的メトリクス

前節で述べたシナリオをオークションシステムに対して入力し,OQFS と OPFS を計測した. そしてソースコードから MM と ICP を計測し, これらの 4 つの計測結果から分析を行っていく.

計測結果を表 8, 表 9, 表 10 に示す.

表 8: メトリクス値の計測データ

	最大値	最小値	平均値	標準偏差
OQFS	67.40	0.00	4.54	11.90
OPFS	45.78	0.00	4.55	9.73
MM	28.00	0.00	4.22	6.61
ICP	48.00	0.00	8.18	12.25

表 9: クラスにおけるメトリクス値の計測データ (6 チーム分)

Comp.	OQFS	OPFS	MM	ICP	OQFS	OPFS	MM	ICP	OQFS	OPFS	MM	ICP
A.C.D.	1.11	0.16	10	21	1.20	0.16	7	16	1.39	0.18	8	17
A.C.	17.26	0.00	12	31	48.91	0.00	11	27	27.83	0.17	9	18
A.M.I.	67.40	3.63	25	40	35.87	3.93	27	41	51.99	5.19	23	40
A.S.	0.41	0.00	6	6	0.09	0.00	1	1	0.11	0.00	1	1
B.D.	1.13	0.11	10	19	0.77	0.11	6	15	1.03	0.15	7	15
B.I.	0.00	0.84	1	1	0.00	0.67	1	1	0.00	0.72	1	1
C.I.	0.00	21.79	6	8	0.00	44.82	0	0	0.00	24.38	0	0
C.B.D.	0.77	0.15	3	6	0.63	0.13	5	11	1.01	0.18	8	16
D.C.E.	0.00	0.00	0	0	0.00	0.00	0	0	0.00	0.00	0	0
D.U.IDE.	0.00	0.00	0	0	0.00	0.01	0	0	0.00	0.01	0	0
E.D.	3.64	0.31	16	27	3.56	0.30	11	24	5.05	0.45	14	28
E.D.D.	3.48	0.26	3	9	2.80	0.24	3	12	4.06	0.32	6	12
E.I.	0.00	17.29	0	0	0.00	14.60	0	0	0.00	15.74	0	0
I.B.E.	0.00	0.00	0	0	0.00	0.00	0	0	0.00	0.01	0	0
I.D.	0.00	1.51	0	0	0.00	1.61	0	0	0.00	1.95	0	0
I.C.E.	0.00	0.01	0	0	0.00	0.00	0	0	0.00	0.01	0	0
I.C.E.	0.00	0.05	0	0	0.00	0.04	0	0	0.00	0.06	0	0
I.E.E.	0.00	0.01	0	0	0.00	0.01	0	0	0.00	0.00	0	0
I.M.E.	0.00	0.01	0	0	0.00	0.01	0	0	0.00	0.02	0	0
M.C.I.	0.00	7.43	0	0	0.00	8.39	0	0	0.00	14.63	0	0
M.I.	0.00	45.78	0	0	0.00	24.09	0	0	0.00	34.77	0	0
M.R.D.	4.69	0.66	11	22	6.17	0.87	10	24	7.54	1.06	4	8
A.C.D.	1.65	0.21	11	22	1.32	0.17	10	24	2.41	0.21	8	17
A.C.	33.70	0.60	20	39	48.22	0.02	13	19	31.59	1.50	23	48
A.M.I.	39.30	7.60	26	45	36.74	4.35	28	44	37.62	5.69	25	37
A.S.	0.13	0.00	1	1	0.09	0.00	1	1	0.13	0.00	1	1
B.D.	0.95	0.16	6	13	0.77	0.11	7	17	1.08	0.23	8	17
B.I.	0.00	2.56	1	1	0.00	0.78	1	1	0.00	1.14	0	0
C.I.	0.00	27.70	1	1	0.00	37.25	0	0	0.00	28.44	4	6
C.B.D.	0.92	0.18	5	10	0.64	0.10	6	14	0.94	0.18	6	11
D.C.E.	0.00	0.01	0	0	0.00	0.00	0	0	0.00	0.01	0	0
D.U.IDE.	0.00	0.01	0	0	0.00	0.01	0	0	0.00	0.01	0	0
E.D.	5.25	0.42	16	34	3.41	0.29	6	12	5.51	0.44	13	27
E.D.D.	5.34	0.35	3	9	2.28	0.21	7	17	4.95	0.35	4	11
E.I.	2.12	14.62	0	0	0.11	18.96	1	1	0.00	15.77	0	0
I.B.E.	0.00	0.00	0	0	0.00	0.00	0	0	0.00	0.01	0	0
I.D.	0.00	2.32	0	0	0.00	1.69	0	0	0.07	2.32	0	0
I.C.E.	0.00	0.01	0	0	0.00	0.01	0	0	0.00	0.01	0	0
I.C.E.	0.00	0.05	0	0	0.00	0.04	0	0	0.00	0.03	0	0
I.E.E.	0.00	0.01	0	0	0.00	0.01	0	0	0.00	0.01	0	0
I.M.E.	0.00	0.02	0	0	0.00	0.01	0	0	0.00	0.00	0	0
M.C.I.	0.00	5.91	0	0	0.00	9.49	0	0	0.00	5.21	0	0
M.I.	0.00	34.61	0	0	0.00	25.56	0	0	0.00	37.17	0	0
M.R.D.	10.63	2.66	5	10	6.41	0.93	6	17	15.70	1.27	8	16

表 10: クラスにおけるメトリクス値の計測データ (1 チーム分)

A.C.D.	1.50	0.36	5
A.C.	34.31	0.60	10
A.M.I.	45.17	5.82	13
A.S.	0.11	0.00	1
B.D.	0.99	0.15	5
B.I.	0.00	0.99	1
C.I.	0.00	28.10	2
C.B.D.	1.05	0.21	4
D.C.E.	0.00	0.00	0
D.U.IDE.	0.00	0.01	0
E.D.	5.24	0.87	6
E.D.D.	3.98	0.72	7
E.I.	0.00	14.93	2
I.B.E.	0.00	0.00	0
I.D.	0.00	2.14	0
I.C.E.	0.00	0.01	0
I.C.E.	0.00	0.02	0
I.E.E.	0.00	0.00	0
I.M.E.	0.00	0.01	0
M.C.I.	0.00	10.20	0
M.I.	0.00	32.66	0
M.R.D.	7.65	2.19	3

これらの計測結果より次の事柄が考察された.

1. OQFS の値が高い時に,MM と ICP もそれぞれ高い値を示す.
2. OQFS の値が高い時には OPFS は低い値を示し,OQFS の値が低いときには OPFS は高い値を示す.
3. MM の値より ICP の値が低い事が存在しない.
4. MM や ICP の値より,OQFS や OPFS は幅広く分散している.

1 については, それぞれのメトリクスの性質を理解していると安易に理解できる. OQFS は計測対象クラスが他のクラスを呼び出す割合を計測している. そして MM は計測対象クラスが他のクラスのメソッドを呼び出す回数を計測している.ICP は計測対象クラスが他のクラスのメソッドを呼び出す回数に引数の重みを加えて計測している. つまり 3 つのメトリクスは計測対象がクラスとメソッドの違いはあるが, 他のクラスやメソッドを呼び出す回数を計測しているという共通点が存在している為と思われる.

2 については, これも 1 と同様にメトリクスの性質から説明できる. OQFS は先程も述べたように計測対象クラスが他のクラスを呼び出す割合を計測している.OPFS は計測対象クラスが他のクラスから呼び出される割合を計測している. ここで 2 つのメトリクスの間にこのような特徴が出るという事は,OQFS の値が高いクラスはシステムの中でメインのクラスで実装されているものと予想でき,OPFS の値が高いクラスはそのメインのクラスの機能を実装し, 補助的なクラスと存在しているものと予想される.

3 についても同様に,MM は計測対象クラスが他のクラスのメソッドを呼び出す回数を計測し,ICP は計測対象クラスが他のクラスのメソッドを呼び出す回数に引数の重みを加えて計測しているので, 引数の重みを加えた分だけ ICP の値が大きくなっている.

4 については, 静的メトリクスと動的メトリクスの計測方法の違いにより説明できる.MM や ICP はソースコードから計測しているので, 実際にそのクラスが呼び出したり呼び出されたりしているかどうかまでは判断できない. つまり, あまり使用される事が予想されない例外処理などもソースコード上では計測してしまう. しかし入力の様子から計測している OQFS や OPFS は, シナリオによって例外処理などが行われない場合は, そのクラスは処理を行っていないと判断するので, それらのクラスの計測値は低くなってしまう. その為に MM や ICP は計測値にあまり相違が見られない.

5.2 動的メトリクスと静的メトリクスの相関

動的メトリクス (OQFS) と静的メトリクス (MM,ICP) の相関係数はそれぞれ 0.79 と 0.75 となった. この値は 1 に近づけば近づくほど相関が高い事からあまり強くは相関していない事が分かる. これらの結果より, 動的メトリクスと静的メトリクスはそれぞれ違った概念で計測をしている事が分かる. この結果はまた Yacoub らの動的メトリクスについての主張の補完となっている事も確認できる. 相関に違いが出る事については, 静的メトリクスは他のクラスやメソッドの呼び出しを, ソースコード上で出現する回数分しか計測できないのに対して, 動的メトリクスは, シナリオに従いそのクラス自体が何度も呼び出されたり全く呼び出されなかったりとシナリオに大きく影響される事から相関があまり強くないという結果が導かれたものと考えられる.

5.3 再利用性

ここでは実験により計測された OQFS を使用する. 判定するクラスの数は 22 個になる. オークションシステムのクラス数は 29 個だがインターフェース部分のクラスを 7 個含む為,

それらは省いた. そしてオークションシステムは 7 チームを対象にしているので, 合計 154 個のクラスが計測対象となる. その 154 個のクラスのうち,OQFS の値が高い 5 クラスと低い 5 クラスを他の 6 つのオークションシステムの同クラスと入れ替えオークションシステムとして機能するかどうかの判定を行った. 入れ替える際に, 一度に複数のクラスを入れ替えると, オークションシステムが機能しなかった場合, どのクラスが原因で機能していないかが判定できない. したがってここでいうクラスの入れ替えは 1 つのクラスを他の 6 つのオークションシステムとの間で行うものと限定した. ここで同クラスと入れ替えるとしているが, なぜそのような事ができるのであろうか. このオークションシステムにおいては, それぞれのクラスの仕様は開発が行われる前に予め決定されており, 同クラスでは同じ機能を実装している事が約束されているからである.

次に入れ替えを行うクラスを示す.

- メトリクス値の高いもの
  - AuctionClient
  - AuctionManagerImpl
- メトリクス値の低いもの
  - BidImpl
  - CategoryImpl
  - DuplicateCategoryException
  - ExhibitImpl

そして入れ替えた実験結果を表 11, 表 12 で示す.

表 11: 動的メトリクス値の上位 5 つ

コンポーネント	コンパイル結果	テスト結果
AuctionClient	0/6	0/6
AuctionClient	4/6	0/6
AuctionManager	4/6	1/6
AuctionManager	5/6	2/6
AuctionManager	3/6	3/6
SUM	16/30	6/30

表 12: 動的メトリクス値の下位 5 つ

コンポーネント	コンパイル結果	テスト結果
BidImpl	6/6	6/6
BidImpl	5/6	5/6
CategoryImpl	6/6	6/6
DuplicateCategoryException	6/6	6/6
DuplicateCategoryException	6/6	6/6
SUM	29/30	29/30

上の結果から,OQFS の値が高い 5 クラスはコンパイルした結果, コンパイルが成功したものは全体で 30 個中 16 個と半分のクラスしかコンパイルに成功していない.OQFS の値が低い 5 クラスについてはコンパイルは 30 個中 29 個とかなり高い確率でコンパイルに成功している.

次にテストであるが, ここで行うテストは入れ替えを行うクラスが実装している機能をシナリオとして入力し不備なく動作するかどうかの判定を行った. ここでのテストの結果は,OQFS の値が高い 5 クラスはコンパイルに通った 16 個中 6 個とまた更に低い値を示している.OQFS の値が低い 5 クラスについてはコンパイルに通った 29 個全てテストにも成功している.

以上のコンパイルとテストの結果から,OQFS の値が高いクラスは再利用性が低い事がわかり,OQFS の値が低いクラスは再利用性が高い事がわかった.

逆に言うと,OPFS の値が低いものは再利用性が低いという事になる.

なぜこの事が言えるかというと , 先程動的メトリクスと静的メトリクスの評価を行なったがその計測結果から明らかである. なぜかというと, それぞれのクラスは, クラスとしての仕様は決定しているが細部までの仕様 (引数の数など) までは決定されていない為, 頻繁にクラスの呼び出しを行うクラスについては, 引数の違い等によりオークションシステムとしての機能を実装するには至っていないものと考察される.

よって, 動的メトリクスの値の大小により再利用の可能性を判定できるものと考察され, 同時にこの結果は Yacoub らの動的メトリクスによる品質特性である再利用性についての考えを補完しているものと思われる.

6 まとめ

本論文では, 複雑度メトリクスを用いたJavaプログラム品質特性 (再利用性) の実験的評価を行った. 動的メトリクスの評価の為に, ホワイトボックステストの手法を用いて, 全てのコンポーネントが実行されるような入力データ (シナリオ) を作成し,Windows Scripting Hostを使用して組織的なテストを行なった.

実験の結果より,Briandらの MM,Leeらの ICP といった静的メトリクスと動的メトリクスとの間には相関関係があまり高くないという結果が得られ, 動的メトリクスは静的な複雑さとは違った複雑さを評価している事が確認できた. そして品質特性としてクラスの再利用性についても動的メトリクスは有効であると確認できた.

今後の課題としては, 再利用性の評価に関して今回は, 同一アプリケーションにおけるコンポーネントの入れ換えを行なう事で再利用性を評価したが, これとは異なる再利用性の評価として, 異なるアプリケーションにおけるコンポーネントの再利用性の評価が考えられる. 今回はJavaでソースコードが書かれたオークションシステムに対して評価実験を行ったが, 同様の実験を他の条件の元において行う事が挙げられる.

謝辞

本論文を作成するにあたり, 常に適切な御指導を賜わりました大阪大学大学院基礎工学研究科情報数理系専攻 井上 克郎 教授に心より深く感謝致します.

本論文の作成において, 適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 楠本 真二 助教授に深く感謝致します.

本論文の作成において, 適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 松下 誠 助手に深く感謝致します.

本論文の作成において, 適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 神谷 年洋 氏に深く感謝致します.

本論文の作成において, 適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 竹原 元康 氏に深く感謝致します.

本論文の作成において, 適切な御指導および御助言を頂きました日本ユニシス株式会社 毛利 幸雄 氏に深く感謝致します.

本論文の作成において, 適切な御指導および御助言を頂きました日本ユニシス株式会社 尾畑 祐一 氏に深く感謝致します.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院基礎工学研究科情報数理系専攻井上研究室の皆様に深く感謝致します．

参考文献

[1] Briand L. C., Daly J. and Wust J.,: “A Unified Framework for Coupling Measurement in Object-Oriented Systems”,IEEE Transaction on Software Engineering, 1999.

[2] Briand L. C., Devanbu P. and Melo W. L.,: “ An Investigation into Coupling Measures for C++”, Proceedings of the 19th International Conference on Software Engineering(ICSE97), 1997.

[3] Basili V. R., Briand L. C. and Melo W. L.,: “A Validation of Object-Oriented Design metrics as Quality Indicators”, IEEE Transaction on Software Engineering, 1996.

[4] Chidamber S. R. and Kemerer C. F.,: “A Metrics Suite for Object Oriented Design”, IEEE Trans. on Software Eng., 1994.

[5] G. Booch ,: “Object-Oriented Analysis and Design with Applications”, The Benjamin/Cummings Publishing Co., 1994.

[6] Hitz M., and Montazeri B.,: “Measuring Product Attributes of Object-Oriented Systems”, Proceedings of the 5-th European Software Engineering Conference(ESEC’95). Barcelona, Spain: Lecture Notes in Computer Science 989, 1995.

[7] Hitz M., and Montazeri B.,: “Measuring Coupling and Cohesion in Object-Oriented Systems”, Proceedings of International Symposium on Applied Corporate Computing, 1995.

[8] 本位田, 青山, 深澤, 中谷,: “オブジェクト指向分析・設計”, 共立出版, 1995.

[9] Kamiya T., Kusumoto S. and Inoue K.,: “ Prediction of Fault-proness at Early Phase in Object-Oriented Development”, The Second IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC’99), 1999.

[10] Lee, Y., Liang B. and Wu S.,: “Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow”, Proceedings of International Conference on Software Qualiy, 1995.

[11] Li, W. and Henry S.,: “Object Oriented Metrics that predict Maintainability”, Journal of Systems and Software, 1993.

[12] Lionel C. Briand, John Daly, Victor Porter, Jurgen Wust ,:“Predicting Fault-Prone Classes with Design Measures in Object-Oriented Sys tems”, Proceeding of the 9th ISSRE, 1998.

[13] Motoyasu, T., Toshihiro K., Shinji K., and Katuro I.,: “Empirical Evaluation of Method Complexity for C++ Program”, IEICE TRANS. INF. & SYST., 2000.

[14] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W.,: “Object-Oriented Modeling and Design”, Prentice Hall, 1991.

[15] 竹原, 神谷, 楠本, 井上, 毛利 ,:“Java を対象とした動的複雑度メトリクスの実験適評価”, 信学技報, 2000.

[16] Victor R. Basili, Lionel C. Briand, and Walce'lio L. Melo ,:“A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Trans. on Software Eng., 1996.

[17] Yacoub S., Ammar H. and Robinson T.,: “Dynamic Metrics for Object Oriented Designs”, Proceedings of the Sixth International Symposium on Software Metrics (METRICS99, 1999.

[18] 山田, 高橋,: “ソフトウェアマネジメントモデル入門 –ソフトウェア品質の可視化と評価法”, 共立出版, 1993.