修士学位論文

題目

オブジェクト指向プログラムのための動的結合メトリクスの評価

指導教官
井上 克郎 教授

報告者
竹原 元康

平成 13 年 2 月 14 日

大阪大学 大学院基礎工学研究科
情報数理系専攻 ソフトウェア科学分野

平成 12 年度 修士学位論文

オブジェクト指向プログラムのための動的結合メトリクスの評価

竹原 元康

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様々なメトリクスが提案されてきている．これまでに提案されている多くのメトリクスは，設計仕様書やソースコードのみから複雑さを評価する，静的なメトリクスであった．しかし最近では，動的な複雑さ (実行時のオブジェクトの振舞い) を評価するメトリクスについての注目が高まってきている．Yacoub らは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を提案している．しかし，その定義上の性質を評価しているに過ぎず，本来評価すべき, (1) 静的な複雑さと動的な複雑さの違いの評価,(2) 品質特性 (保守性, 理解容易性, 再利用性, エラーの伝播と発生可能性) については, 議論されていない. 本研究では，上述した (1), (2) を実施するための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プログラム開発プロセスから, データ収集を行い, 収集したデータを用いて, 静的メトリクスの１つである CBO と Yacoub らのメトリクス (OQFS,OPFS) を実際のJava プログラムに適用した. 実験の結果，CBO より OQFS の方がソフトウェアの開発プロセスにおいて発生したエラー数, またその修正時間との相関が高くなり, 保守性, 理解容易性, エラーの伝播と発生可能性について静的メトリクスより有効であることが確認できた.

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

近年，オブジェクト指向方法論に基づくソフトウェア開発が活発に行われるようになってきている．オブジェクト指向開発においても，ソフトウェアの開発管理を効果的に行うためには，開発の各工程において適切なメトリクスを使用して品質や進捗状況を定量的に評価する必要がある．特に，早期の開発工程でソフトウェアの品質を評価することは，後の開発工程での工数割当や品質管理における指標として非常に有益なものとなる．従来の手続き的な開発方法で開発されたソフトウェアに対するメトリクスでは不十分であるため，オブジェクト指向で開発されたソフトウェアに対するメトリクスが数多く提案されてきている．

オブジェクト指向ソフトウェアに対する代表的なメトリクスとして，Chidamber らの複雑度メトリクス (以降，C&K メトリクス) がある [4]．彼らは 6 種類のメトリクスを定義し，それぞれのメトリクスでクラス間の結合や継承関係等の複雑さを計測することを提案している．C&K メトリクスは，設計仕様書やソースコードの情報のみから複雑度を計測する静的な複雑度メトリクスである．一方，最近では，実行時のソフトウェアの振舞いを評価するための，動的な複雑度メトリクスについての注目が高まっている．

Yacoub らは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を提案した [17]．彼らのメトリクスはある特定の入力データシナリオに対する実行時のクラス間メッセージの送受信回数や状態遷移の複雑さを評価するものである．しかし，文献 [17] では，その定義上の性質を評価しているに過ぎず，本来評価すべき, (1) 静的な複雑さと動的な複雑さの違いの評価,(2) 品質特性 (保守性, 理解容易性, 再利用性, エラーの伝播と発生可能性) については, 議論されていない.

本研究では，上述した (1), (2) を実施するためのソフトウェア工学実験を計画し, 本研究では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したデータを用いることで，Yacoub らのメトリクスの評価を行うことを目的とする．具体的には，Yacoub らが提案した動的メトリクスのである OQFS, OPFS(アプリケーションの実行シナリオに基づいて，クラス間の結合関係を評価したもの) を評価する．先ず，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から収集したデータを用いて OQFS,OPFS と Chidamber らによって提案されている CBO[4] との比較を行い，動的なメトリクスと静的なメトリクスとの相違点を考察する．次に，OQFS とアプリケーションの開発工程において収集されたエラー数との関係を評価し，結果の考察を行う．

結果として, エラー数とエラー修正に要した時間に関して,CBO より OQFS との相関が高くなることが確認できた. また,CBO と OQFS を使用することで, ソースコード上での結合数が同様でもより複雑なクラスを判定できることが確認できた.

また,OQFS と OPFS の相関結果から, 再利用性の高いクラスは OPFS の価が高くなるこ

とが確認できた.

さらに, 保守性, 理解容易性, エラーの伝播と発生可能性について静的メトリクスより有効であることが確認できた.

以下, 2.,3. では Chidamber らの静的メトリクスと Yacoub らの動的メトリクスの説明を行い，残されている課題について述べる．4. では, 本研究で行った評価実験について述べる．5. では実験で得られたデータの分析を行い，最後に 6. 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.

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

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].

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 JVMPI

本研究においてメトリクスを計測する際に JVM に付随の JVMPI(Java Virtual Machine Profiler Interface) を使用した. JVMPI について説明する.

2.6.1 概要

JVMPI は, Java Virtual Machine とプロセス中のプロファイラエージェントとの間の双方向の関数呼び出しインタフェースである.Virtual Machine は, プロファイラエージェントに, ヒープ割り当て, スレッドの開始などに対応するさまざまなイベントを通知します. 一方, プロファイラエージェントは, JVMPI を使ってより多くの情報を制御, および要求します. たとえば, プロファイラエージェントは, プロファイラフロントエンドの必要に基づき, 特定のイベント通知をオンまたはオフすることができます.

プロファイラフロントエンドは, 必ずしもプロファイラエージェントと同じプロセスで実行される必要はありません. プロファイラフロントエンドが, 同じマシン上の別のプロセスにあったり, ネットワークで接続されたリモートマシン上のプロセスにある場合もあります. JVMPI は, 標準のワイヤプロトコルを指定しません. ツールベンダーは, ほかのプロファイラフロントエンドの必要に応じてワイヤプロトコルを設計できます.

JVMPI を基にしたプロファイリングツールを使用することにより, 多量のメモリが割り当てられている場所, CPU に負荷のかかるホットスポット, 不必要なオブジェクトの保存, モニターの競合など, パフォーマンス全般の分析に役立つ多くの情報が取得できます.

JVMPI によって, 部分的なプロファイリングもサポートされています. ユーザは, JVM

が動作する特定の期間を指定してアプリケーションのプロファイリングを行なったり, 特定の種類のプロファイリング情報を選んで取得することもできます.

2.6.2 初期設定

ユーザは, コマンド行オプションを使って Java Virtual Machine に, プロファイラエージェントの名前およびプロファイラエージェントに対するオプションを指定できます. たとえば, ユーザが次のように指定したとします.

java -Xrunmyprofiler:heapdump=on,file=log.txt ToBeProfiledClass

VM は, Java の次のライブラリディレクトリで myprofiler というプロファイラエージェントライブラリを探します.

Win32 の場合, $ JAVA_HOME\hbinE\myprofiler.dll

Java ライブラリディレクトリでライブラリが見つからない場合は, それぞれのプラットフォームの通常のライブラリ検索方法に従って, ライブラリの検索が続けられます. Win32 では, VM はカレントディレクトリ, Windows システムディレクトリ, および PATH 環境変数に指定されたディレクトリを検索する VM はプロファイラエージェントライブラリをロードし, エントリポイントを探します.

jint JNICALL JVM_OnLoad(JavaVM *jvm, char *options, void *reserved);

VM は, JavaVM インスタンスへのポインタを第 1 引数, 文字列 "heapdump=on,file=log.txt" を第 2 引数として渡して, JVM_OnLoad 関数を呼び出します. JVM_OnLoad への第 3 引数は予約されており, NULL に設定されます. 成功した場合は, JVM_OnLoad 関数は JNI_OK を返します. なんらかの理由で JVM_OnLoad 関数の実行が失敗した場合は, JNI_ERR を返します.

2.6.3 イベントの通知

VM は, JVMPI_Event データ構造体を引数として NotifyEvent を呼び出すことにより, イベントを送信します. サポートするイベントは次のとおりです.

- メソッドに入る, およびメソッドから出る
- オブジェクトの割り当て, 移動, および解放
- ヒープ領域の作成および削除
- GC の開始および終了
- JNI グローバル参照の割り当ておよび解放

- JNI グローバル弱参照の割り当ておよび解放
- コンパイルされたメソッドのロードおよびアンロード
- スレッドの開始および終了
- クラスファイルデータの設置準備完了
- クラスのロードおよびアンロード
- 競合する Java モニターに入るのを待つ, 入る, および Java モニターから出る
- 競合する raw モニターに入るのを待つ, 入る, および raw モニターから出る
- Java モニターに対し待機するおよび待機を終了する
- モニターダンプ
- ヒープダンプ
- オブジェクトダンプ
- プロファイリングデータのダンプ要求またはリセット要求
- Java Virtual Machine の初期化およびシャットダウン

2.6.4 イベント

ここでは,動的メトリクスを計測する際に使用したイベントについて使用する.

JVMPI_EVENT_CLASS_LOAD

VM にクラスがロードされるか, またはプロファイラエージェントが RequestEvent の呼び出しの発行によって JVMPI_EVENT_CLASS_LOAD イベントを要求するときに送信されます. 後者の場合は, イベント型に JVMPI_REQUESTED_EVENT ビットが設定されます. このイベントは, GC が無効な状態で発行されます. GC は, NotifyEvent の復帰後に再度有効になります.

```
struct {
    char *class_name;
    char *source_name;
    jint num_interfaces;
    jint num_methods;
    JVMPI_Method *methods;
```

```
    jint num_static_fields;
    JVMPI_Field *statics;
    jint num_instance_fields;
    JVMPI_Field *instances;
    jobjectID class_id;
} class_load;
```

内容:

class_name - ロードされているクラス名
source_name - クラスを定義するソースファイル名
num_interfaces - このクラスによって実装されるインタフェースの数
methods - クラス内に定義されたメソッド
num_static_fields - このクラス内で定義された static フィールドの数
statics - このクラス内で定義された static フィールド
num_instance_fields - このクラスで定義されたインスタンスフィールドの数
instances - このクラスで定義されたインスタンスフィールド
class_id - クラスオブジェクト ID

注: クラス ID は, クラスオブジェクトの ID で, JVMPI_EVENT_OBJECT_MOVE の着信時に変更されます.

JVMPI_EVENT_METHOD_ENTRY

メソッドに入るときに送信されます. このイベントはメソッドの呼び出し対象であるオブジェクトの jobjectID を送信しません.

```
struct {
    jmethodID method_id;
} method;
```

内容:

method_id - 入るメソッド

JVMPI_EVENT_METHOD_EXIT

メソッドの終了時に送信されます. メソッドの終了とは, 通常に終了した場合か, または処理されない例外があった場合を指します.

```
struct {
```

```
    jmethodID method_id;
} method;
```

内容:

method_id - 出ていくメソッド

JVMPI_EVENT_JVM_SHUT_DOWN

VM がシャットダウンしているときに VM によって送信されます. 通常, プロファイラは, プロファイリングデータを保存することによって対応します. イベント特有データはありません.

3 複雑度メトリクス

3.1 動的メトリクスと静的メトリクス

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．

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

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

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.

図 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. 図 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.

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

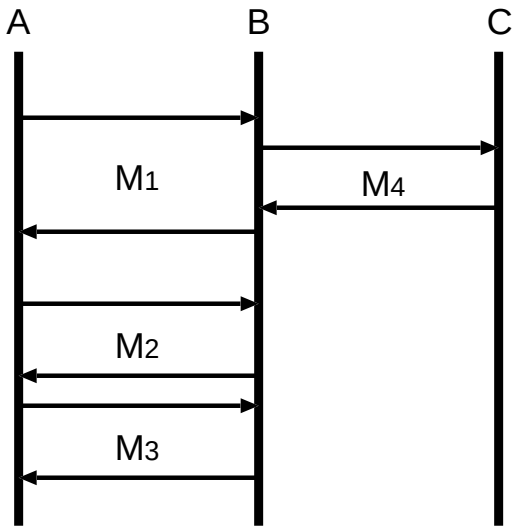

図 1: 動的メトリクスの計測例

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

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．

4 評価実験

4.1 実験計画

実験計画について以下のようなことが言われている.

ソフトウェア工学の分野で数多くの手法やツールが提案されてきているが，その有効性についての具体的な証拠はほとんど収集されていない [2].

以上のような事から, 実際のソフトウェア開発現場や研究機関において様々な状況 (コンテキスト) を設定し，理論や技術の適用実験 (ソフトウェア実験) を行い，その妥当性や有効性などを検証する必要がある.

そこで, 本研究では以下のように実験計画を行う.

- 保守性，理解容易性
  コンポーネントを作成する際に，発生したエラーを修正するのに要した時間を利用して評価する

- 再利用性
  同一仕様を基に作成された複数のクラスを交換し，動作の可否を行うことによって評価する

- エラーの伝播と発生可能性
  あるコンポーネントにエラーが存在し，調査した結果，他のコンポーネントが原因
  →コンポーネントを作成する際に，発生したエラー数によって評価する

4.2 実験目的

我々の目的は，文献 [17] で行われていなかった Yacoub らの動的メトリクスと静的メトリクスの関係の評価と動的メトリクスが 4 つの品質特性 (保守性，理解容易性，再利用性，エラーの伝播と発生可能性) を評価するのに有効であるかどうかを実験的に確認することである．上記の 4 つの品質特性を以下の方法で評価を行う.

- 保守性, 理解容易性 - エラー修正時間

- 再利用性 - クラスの入れ換え

- エラーの伝搬と発生可能性 - エラー数

本稿では，保守性, 理解容易性, エラーの伝播と発生可能性ついての評価を行う.

また, 動的メトリクスと静的メトリクス間にどのような関係があるかも調べる.

具体的には, 先ず, 実験により作成された複数のクラスに対して, 動的メトリクスの OQFS と静的メトリクスの CBO を計測し，それらの間の相関関係を調べる．次に，各クラスで発生したエラー数と OQFS の間の関係を調べる.

また,OQFS と OPFS の間にどのような関係があるか, その関係が品質特性にどのような影響があるのかを考察する.

4.3 実験概要

本研究で使用するデータは, 日本ユニシス株式会社の 2000 年 7 月に行われたJava プログラム開発演習から収集した. 研修生 (被験者) は演習の前に, オブジェクト指向開発について講習を受けている. この演習では 1 チーム 4〜5 名からなるチーム開発が行なわれ, 全 36 チームが独立して同じシステムの開発を行った.

開発されたシステムはオークションシステムである．各グループのメンバーがオークションシステムを構築する上で必要なコンポーネント (表 7 参照) を作成する. これらのコンポーネントに関しては, あらかじめ仕様がインストラクタによって定義されている.

各グループは次の 1〜10 の流れで開発を行った.

1. 仕様の理解
2. 仕様レビュー
3. 設計
4. 設計レビュー
5. テストケース作成
6. テストケースレビュー
7. コーディング
8. コードレビュー
9. テスト
10. デバッグ

プログラミング言語はJava であり, 処理系は JDK1.2.2 である. 開発されたオークションシステムはインストラクタによりテストされ要求仕様を満たすことが確認されている．システムの規模は 1200 行程度である.

オークションシステムは以下の機能を持つ.

1. 会員登録処理:

   実際に出品あるいは入札を行うために会員登録を行うための処理である.

2. 出品処理:

   ユーザが売却したい品物をオークション（競売）にかけるために行う処理である.

3. 入札処理:

   客が出品物の入札価格を書いて提出するための処理である.

4.4 データ収集方法

以下の 2 項目のデータを収集した.

1. Java プログラムのソースコード

2. テスト工程とレビュー工程におけるエラーデータ

1. に関して, 被験者はそれぞれ, 割り当てられたパーソナルコンピュータ (PC) 上で開発作業を行なう. 各 PC およびサーバーは同一のネットワークに接続されている. サーバーは AM10:00〜PM20:00 までの間, 1 時間毎に被験者の作業ディレクトリをバックアップすることで自動的にプログラムソースファイルを収集する. これを用いてメトリクスの値を計測する.

2. に関して, 被験者には課題の要求仕様やデータフォーマットなどの資料が配布されている. テストは, テストケース作成工程において作成されたテストケースに基づいて行われた. 開発プロセスや, エラーデータの報告には, 紙媒体を使用した報告書に被験者が記入することにする.

報告書の記入項目は以下の通りである.

- 開発コンポーネント
- エラーの位置と内容
- エラー修正開始日時, 終了日時

4.5 収集データ

主に次の 2 種類のデータを収集した.

(1) プログラムコード：メトリクス値を計測するためのデータ.

(2) エラーデータ：主に，設計レビュー，コードレビュー，テストにおいて発見されたエラーと各エラーの修正に要した時間に関するデータ.

被験者である 36 チームのソースコードのうち仕様を満たしている 7 チーム分を分析対象データとした．判定方法であるが, 実際にオークションシステムを動作させ作成したシナリオ通りに動作すれば分析対象とすることにしている.

収集された各クラスのソースコードの行数は, 50 行程度のものから 500 行程度のものまで幅広く分散している．これには空白行や, コメント行も含まれる. また, これらの 154 個のクラスから収集されたエラーデータの総数は, 80 個である．これらの分析対象データをもとに, 実際にメトリクスを計測し評価していく.

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

4.6 使用したシナリオ

今回の実験において使用したシナリオについて説明を行う.

すべてのシナリオについて, ユーザを 10 人用意する. その 10 人が会員登録, 出品, 入札といった処理を繰り返し行う.

シナリオは, 全部で 30 種類用意した. このシナリオの中には, 会員登録だけを行うシナリオから, 登録する時の入力ミスを処理する例外処理を行うシナリオまで, 様々なパターンのシ

ナリオを用意した.

シナリオの例を以下に示す.

A. 会員登録→ B. 会員登録→ A. 出品→ C. 会員登録→ B. 出品→ C. 入札……

5 分析

5.1 計測結果

前節で述べた実験データに対して，シナリオを入力し，OQFS,OPFS と CBO を計測した．この実験には 30 個のシナリオを用意して実行を行った．これらの計測結果を以降に示し，考察していく．

各コンポーネントの計測結果を表 8,9 に示す．

結果より次の事が観察される．

1. CBO に比べて OPFS,OQFS の方が幅広く分散している
2. OQFS の計測値が高い時に OPFS の計測値は低く,OPFS の計測値が高いときに OQFS の値は高い

1. に関して,CBO はソースコードから計測されているので, 実行時に使用されないクラスでも, ソースコード上で他のクラスを使用していれば, それは結合として計測される. それに対して OQFS,OPFS は実際に想定されるシナリオを入力データとして与えるため, 実行時にあまり使用されないクラス (例えば例外処理クラス) は計測値が低くなる.CBO は, 実行をさせずに計測しているので, どのクラスがどの程度使用されるかが全く分からないため, クラス間での計測値にあまり相違が出てこない.

2.OQFS は他のクラスを呼び出している割合を計測しているのに対し,OPFS は呼び出される割合を計測している. つまり OQFS の値が高いクラスは, システムにおいて中心的な機能を持っており, また OPFS の値が高いクラスは, 多く呼び出されているということから, 補助的な機能をもったクラスであると考えることができる. 今回の実験データでは,OQFS と OPFS の計測値の違いが明確であるため, 各クラスがそのシステムにおいてどのような役割を果たしているかを推測する指針とすることができる.

表 8: 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
CBO	13	0	3.06	3.85

表 9: 計測データ

Comp.	OQFS	OPFS	CBO	OQFS	OPFS	CBO	OQFS	OPFS	CBO	OQFS	OPFS	CBO
A.C.D.	1.11	0.16	7	1.20	0.16	7	1.39	0.18	8	1.50	0.36	5
A.C.	17.26	0.00	11	48.91	0.00	10	27.83	0.17	10	34.31	0.60	10
A.M.I.	67.40	3.63	13	35.87	3.93	13	51.99	5.19	13	45.17	5.82	13
A.S.	0.41	0.00	6	0.09	0.00	1	0.11	0.00	1	0.11	0.00	1
B.D.	1.13	0.11	7	0.77	0.11	7	1.03	0.15	7	0.99	0.15	5
B.I.	0.00	0.84	1	0.00	0.67	1	0.00	0.72	1	0.00	0.99	1
C.I.	0.00	21.79	2	0.00	44.82	2	0.00	24.38	2	0.00	28.10	2
C.B.D.	0.77	0.15	5	0.63	0.13	6	1.01	0.18	6	1.05	0.21	4
D.C.E.	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0
D.U.IDE.	0.00	0.00	0	0.00	0.01	0	0.00	0.01	0	0.00	0.01	0
E.D.	3.64	0.31	7	3.56	0.30	7	5.05	0.45	7	5.24	0.87	6
E.D.D.	3.48	0.26	6	2.80	0.24	8	4.06	0.32	7	3.98	0.72	7
E.I.	0.00	17.29	2	0.00	14.60	2	0.00	15.74	2	0.00	14.93	2
I.B.E.	0.00	0.00	0	0.00	0.00	0	0.00	0.01	0	0.00	0.00	0
I.D.	0.00	1.51	0	0.00	1.61	0	0.00	1.95	0	0.00	2.14	0
I.C.E.	0.00	0.01	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01	0
I.C.E.	0.00	0.05	0	0.00	0.04	0	0.00	0.06	0	0.00	0.02	0
I.E.E.	0.00	0.01	0	0.00	0.01	0	0.00	0.00	0	0.00	0.00	0
I.M.E.	0.00	0.01	0	0.00	0.01	0	0.00	0.02	0	0.00	0.01	0
M.C.I.	0.00	7.43	0	0.00	8.39	0	0.00	14.63	0	0.00	10.20	0
M.I.	0.00	45.78	0	0.00	24.09	0	0.00	34.77	0	0.00	32.66	0
M.R.D.	4.69	0.66	5	6.17	0.87	5	7.54	1.06	5	7.65	2.19	3
A.C.D.	1.65	0.21	8	1.32	0.17	7	2.41	0.21	8			
A.C.	33.70	0.60	9	48.22	0.02	10	31.59	1.50	10			
A.M.I.	39.30	7.60	13	36.74	4.35	13	37.62	5.69	13			
A.S.	0.13	0.00	1	0.09	0.00	1	0.13	0.00	1			
B.D.	0.95	0.16	8	0.77	0.11	8	1.08	0.23	7			
B.I.	0.00	2.56	1	0.00	0.78	1	0.00	1.14	1			
C.I.	0.00	27.70	2	0.00	37.25	2	0.00	28.44	2			
C.B.D.	0.92	0.18	6	0.64	0.10	5	0.94	0.18	5			
D.C.E.	0.00	0.01	0	0.00	0.00	0	0.00	0.01	0			
D.U.IDE.	0.00	0.01	0	0.00	0.01	0	0.00	0.01	0			
E.D.	5.25	0.42	8	3.41	0.29	7	5.51	0.44	8			
E.D.D.	5.34	0.35	6	2.28	0.21	6	4.95	0.35	4			
E.I.	2.12	14.62	2	0.11	18.96	2	0.00	15.77	2			
I.B.E.	0.00	0.00	0	0.00	0.00	0	0.00	0.01	0			
I.D.	0.00	2.32	0	0.00	1.69	0	0.07	2.32	0			
I.C.E.	0.00	0.01	0	0.00	0.01	0	0.00	0.01	0			
I.C.E.	0.00	0.05	0	0.00	0.04	0	0.00	0.03	0			
I.E.E.	0.00	0.01	0	0.00	0.01	0	0.00	0.01	0			
I.M.E.	0.00	0.02	0	0.00	0.01	0	0.00	0.00	0			
M.C.I.	0.00	5.91	0	0.00	9.49	0	0.00	5.21	0			
M.I.	0.00	34.61	0	0.00	25.56	0	0.00	37.17	0			
M.R.D.	10.63	2.66	6	6.41	0.93	5	15.70	1.27	5			

表 10: 各メトリクスとエラー数, 修正時間との相関係数

	OQFS	OPFS	CBO
エラー数	0.558	-0.41	0.515
エラー修正時間	0.561	0.49	0.433

表 8 では，OQFS と CBO の計測結果に対して，最大値，最小値，平均値，標準偏差を算出した．CBO は離散値であるのに対し，OQFS は連続値となっている．特に OQFS の値が高かったのが，AuctionClient クラスと AuctionManagerImpl クラスであった．このことから，このアプリケーションでは，AuctionClient クラスと AuctionManagerImpl クラスが他のオブジェクトとメッセージの送受信が多いことがわかる．このことより，この 2 つのクラスがオークションシステムにおいて中心となるクラスであることが分かる．

5.2 動的メトリクスと静的メトリクスの相関

動的メトリクスと静的メトリクスの相関係数は 0.73 であり，あまり強く相関していないことが分かる．この結果は，動的メトリクスと静的メトリクスとは異なった観点を評価しているという Yacoub らの主張を補完するものになっている．静的なメトリクスは，アプリケーションが実行された際のオブジェクト間の頻繁なメッセージの送受信が考慮されていないため，このような結果になったと考えられる．

5.3 エラー数, 修正時間との相関

表 10 では，各クラスに対する動的メトリクス，静的メトリクスのそれぞれの値とアプリケーションを開発する際に発生したエラー数とそのエラーを修正するのに要した時間との相関係数を示している．

この結果を見ると，それぞれの相関係数は，エラー数よりエラーの修正時間に要した時間との相関が高くなっている. また, エラー数, 修正時間のどちらに関しても CBO より OQFS の方が相関が高くなっている. エラー数よりエラーの修正時間で重み付けした方がより差がはっきりしていることがわかる．これは, 同じ 1 つのエラーでもその修正時間によって, エラーの難易度が異なっているため差が現れたと考えられる. つまり, エラーの修正時間が長ければそのエラーはより複雑なものと考えることができる. よって, エラー数との相関より修正時間との相関のほうがよりそのクラスの複雑度を反映していると考えることができるため,CBO より OQFS のほうがよりクラスの複雑度を評価していると考えることができる．

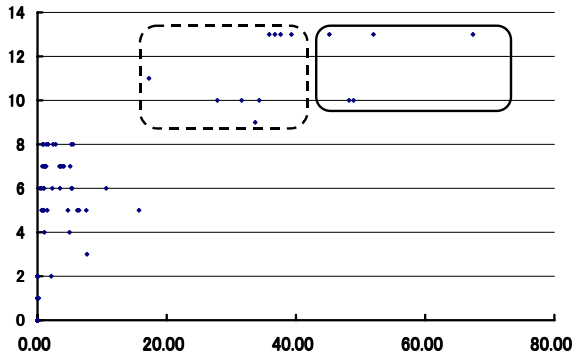

図 2: OQFS vs. CBO

5.4 メトリクスの利用方法

各クラスの静的メトリクスの値と動的メトリクスの値を散布図にしたものを図 2 に示す. グラフの左下の方は，静的メトリクスの値と動的メトリクスの値が共に小さく，静的な結合が少ないクラスが，実際の実行シナリオの上でもあまりメッセージの送受信を行っていないということを示す．また，逆に右上の方は，静的なメトリクスの値と動的なメトリクスの値が共に大きく，メッセージの送受信が多く行われていることが分かる．CBO の値が 10,13 の点はそれぞれ同じクラスである. しかし, それぞれのクラスで OQFS の値は大きく異なっている. そこで, 実際にグラフの実線部分と破線部分でエラー修正に要した時間の平均値を求めてみると, 実線で囲まれた部分の方が約 2.5 倍大きくなっていた.

このような散布図から，設計段階での静的な結合度とシナリオの上で実行させたときの結合度を見ることによって，どのクラスがどれぐらいの割合でアクセスされるかを判断する指標にすることができる．動的なメトリクスを用いることで, 更に絞りこむことが可能となる. アプリケーションの保守，管理に役立つことになると考えられる.

5.5 クラスの Error-Proneness の評価

一般的にアプリケーションの品質を評価するときに，1 つのメトリクスのみを用いるというのは現実的ではなく，複数のメトリクスを合わせて評価することが必要である．今回の実験では，動的メトリクス，静的メトリクスをそれぞれ 1 つずつ使用して評価したが，複数のメトリクスを合わせて評価してみる必要がある．合わせて評価することで，メトリクスの様々な側面を評価することが可能である．例えば，あるクラスにエラーが存在するかしないかを予測する方法として，ロジスティック回帰分析を用いた手法が提案されている [3][9]．こ

の手法はあるクラスにエラーが存在するかどうかを予測するときに，複数のメトリクスを指標として使用し，分析する方法である．文献 [9] では，これらのメトリクスに静的メトリクスしか使用されていなかったが，動的メトリクスを指標の 1 つに加えることで，エラー予測の精度が上昇すると考えられる．精度が上昇すれば開発工程の期間の短縮にもつながり，アプリケーションの保守，管理にも役立つと考えられる．

6　まとめ

本論文では，Yacoub らの提案したオブジェクト指向ソフトウェアに対する動的複雑度メトリクスの有効性についての実験的評価を行った．実験の結果，Chidamber らの静的メトリクスとの間の相関関係はあまり高くなく，静的な複雑さとは異なる観点の複雑さを評価していることが確認できた．また, 以下の品質特性について静的メトリクスより動的メトリクスより有効であることが確認できた.

- 保守性，理解容易性
- 再利用性
- エラーの伝播と発生可能性

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.

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, Vol. 25, No. 1, pp. 91-121 (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), Boston, pp. 412-421(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, Vol. 20, No. 22, pp. 751-761(1996).

[4] Chidamber S. R. and Kemerer C. F.: “A Metrics Suite for Object Oriented Design”, IEEE Trans. on Software Eng., Vol. 20, No. 6, pp. 476-493 (1994).

[5] G. Booch: “Object-Oriented Analysis and Design with Applications”, 2nd Edition, The Benjamin/Cummings Publishing Co., Inc (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, Springer-Verlag, pp. 124-136 (1995).

[7] Hitz M., and Montazeri B.: “Measuring Coupling and Cohesion in Object-Oriented Systems”, Proceedings of International Symposium on Applied Corporate Computing, Monterrey, Mexico, (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), Saint Malo, pp.253-258 (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, Maribor, Slovenia (1995).

[11] Li, W. and Henry S.: “Object Oriented Metrics that predict Maintainability”, Journal of Systems and Software, Vol. 23, No.2, pp. 111-122 (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, pp.334-343(1998).

[13] Motoyasu, T., Toshihiro K., Shinji K., and Katuro I.,“Empirical Evaluation of Method Complexity for C++ Program”, IEICE TRANS. INF. & SYST., VOL.E83-D, NO.8 Aug 2000, pp1698-1700.

[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,pp17-24.

[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., Vol 22, No 10, pp.751-761(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), Boca Raton, Florida USA, pp. 50-61 (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).