JAVA を対象とした動的複雑度メトリクスの実験的評価

竹原 元康† 神谷 年洋† 楠本 真二† 井上 克郎†‡ 毛利 幸雄∗

†大阪大学大学院基礎工学研究科情報数理系専攻

〒560-8531 大阪府豊中市待兼山町 1-3

Phone: 06-6850-6571 Fax: 06-6850-6574

‡奈良先端科学技術大学院大学情報科学研究科, ∗日本ユニシス株式会社

E-mail: takehara@ics.es.osaka-u.ac.jp

あらまし

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．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，本来評価すべき開発工数やエラー数等との関係の評価はされていない．本研究では，Yacoub らのメトリクスを実際の JAVA プログラムに適用することで，有効性の評価，適用上の問題点について考察する．

キーワード オブジェクト指向開発，複雑度，メトリクス，JAVA，実験的評価

Empirical evaluation of the dynamic complexity metrics for JAVA

Motoyasu Takehara†, Toshihiro Kamiya†, Shinji Kusumoto†, Katsuro Inoue†‡ and Yukio Mohri∗

†Graduate School of Engineering Science, Osaka University

1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

Phone: +81-6-6850-6571 Fax: +81-6-6850-6574

‡Graduate School of Inforamtion Science, Nara Institute of Science and Technology, ∗Nihon Unisys Corporation

E-mail: takehara@ics.es.osaka-u.ac.jp

Abstract

In order to evaluate complexity of object-oriented software, many metrics have been proposed. Most of them are aimed to evaluate the static complexity. Nowadays, it is widely noticed the importance to evaluate dynamic complexity of the object-oriented software. Yacoub et al. proposed the dynamic complexity metrics for object-oriented software. However, they only compared the differences between

their metrics and the conventional static complexity metrics about their properties from the definition. From the practical point of view, it is necessary to evaluate the relationship among the metrics and the several properties (e.g. number of faults) of the software development. This paper empirically evaluates the usefulness of the Yacoub's metrics using the data collected from actual object-oriented software development.
Key words Object-oriented development, Complexity, Metrics, JAVA, Empirical evaluation

1 はじめに

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 [10]．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 [10] では，その定義上の性質を評価しているに過ぎず，本来評価すべき開発工数やエラー数等との関係は評価されていない．

本研究では実際のソフトウェア開発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 1 つである OQFS(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 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．

以下，2. 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．3. では，本研究で行った評価実験について述べる．4. では実験で得られたデータの分析を行い，5. では分析結果の考察を行う．最後に 6. でまとめと今後の課題について述べる．

2 複雑度メトリクス

2.1 静的メトリクスと動的メトリクス

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 [5]．

(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．

2.2 C&K メトリクス

今日までに，静的なメトリクスとして，結合，凝集，複雑度の様々なメトリクスがオブジェクト指向ソフトウェアに対するメトリクスとして提案されている．その中でも代表的なものに，

Chidamber と Kemerer が提案したメトリクスがある．Chidamber と Kemerer はオブジェクト指向ソフトウェアに対してこれらの観点を評価するための 6 つのメトリクスを定義した．これらの定義を以下に示す. なお, 計測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ッド数．クラスのメソッドがどれも同じくらいの複雑さであると仮定できれば, クラスのメソッド数となる．

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, 計測対象クラスの木の中での深さとなる．

NOC(Number Of Children) ：

子クラスの数．計測対象クラスから直接派生しているクラスの数となる．

CBO(Coupling Between Object class) ：

クラス間の結合．計測対象クラスが「結合」しているクラスの数．ここで結合とは，計測対象クラスが他のクラスの属性やメソッドを用いていることを意味する．

RFC(Reference For a Class) ：

クラスに対する参照．計測対象クラスのメソッドの集合と, 各メソッドで呼び出す他のクラスのメソッドの集合の和集合の要素数となる.

LCOM(Lack of Cohesion in Methods) ：

メソッドの凝集の欠如．あるクラスのメソッ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 より小さくなったときは 0 と定義する.

2.3 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．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 [10] 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．

2.3.1 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.

2.3.2 $EOC_x(o_i, o_j)$ 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．

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

ここで，K はオブジェクトの集合である．

2.3.3 $IOCx(o_i,o_j)$ 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．

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

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．

2.3.4 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．

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

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

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

表 1: 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

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

2.4 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 1 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 評価実験

3.1 実験目的

我々の目的は，文献 [10] 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．本稿では，動的メトリクスと静的メトリクス間の関係と動的メトリクスによるエラーの伝播と発生可能性についての評価を行う．

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．

3.2 実験概要

実験データは, 2000 年 7 月に行われたある企業の新人研修における 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 2 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 の作業を行った.
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.

3.3 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.

表 2: 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
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
InformationDialog	情報表示ダイアログ
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

(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 24 チーム分を分析対象データとした．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 137 個のクラスから収集されたエラーデータの総数は, 258 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.

4 分析

前節で述べた実験データに対して，1 つのシナリオを入力し，OQFS と CBO を計測した．このシナリオは,「オークションシステムのユーザ 3 人がまず会員登録を行い，そのうち 2 人が何らかの商品を出品し，この 2 人が出品した商品を残りの 1 人が入札する」というものである.

表 3: 計測データ

	最大値	最小値	平均値	標準偏差
OQFS	70.7	0.27	14.9	16.0
CBO	19	1	6.97	5.54

これらの計測結果を以降に示し，考察していく.

4.1 計測結果

表 4: エラー数との相関係数

	OQFS	CBO
エラー数	0.51	0.49

表 3 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 Auction-

ManagerImpl 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．

4.2 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 0.59 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．

4.3 エラー数との相関

表 4 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，ほぼ同じになっている．今回の実験では，メトリクスとエラー数との相関はあまりみられなかった．つまり，動的メトリクス，静的メトリクスのそれぞれの値が増加，減少してもエラー数はそれほど変化していないということが分かる．その原因としては，今回の実験では，動的メトリクスを計測する際に，シナリオを 1 つしか用いていないため，差があまり現れなかったことが考えられる．従って，さらにシナリオを増やし，かつ 1 個あたりの実行シナリオを長いものにして，再実験を行う必要がある．

4.4 個々のクラス特性

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 1 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．このグラフで特徴的なのは，静的なメトリクスの値が小さいにもかかわらず，動的なメトリクスの値が大きくなっているクラスが存在することである (グラフの左上部分)．このようなクラスは，実行シナリオ上では，頻繁にメッセージの送受信をしていることが分かる．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．動的なメトリクスは，このような各クラス別の特性を見る上でも重要な役割を果たしている．各クラス別の特性を知ることは，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．

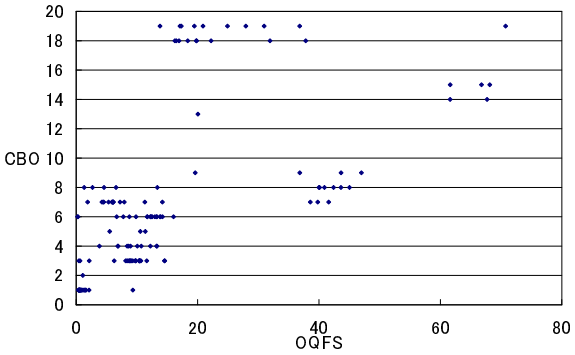

図 1: OQFS vs. CBO

5 考察

ここでは，今後どのように実験を行うか，またどのような点を実験で評価する必要があるかを述べる．今回の実験では，計測しきれていないものが非常に多くあり，それを計測するにはどのようにすれば良いかを，シナリオをさらに増やすという観点，クラスの規模ごとの比較，他の静的メトリクスとの比較，他のメトリクス

との合わせての比較という点で述べる．

5.1 シナリオの増加

今回の実験では，シナリオ 1 つに対して実験を行った．シナリオが 1 つであったために，シナリオの確率は導入されていない．今後の実験では，シナリオを更に増やし，かつある程度長いシナリオで計測する必要がある．また，全てのクラスが実行時に使用されるようなシナリオが含まれることも望まれる．もし，実行時に使用されないクラスがあれば，そのクラスの動的メトリクスは計測されないので，評価することができない．

シナリオの確率は，設計段階において考えられるユースケースに基づいて設定する必要がある．この動的メトリクスは，シナリオ確率の影響を大きく受けるので，設計段階でどのようなユースケースがあるのかを詳細に考える必要がある．ユースケースがきちんと考えられていないと，後に動的メトリクスを計測してもシナリオの確率によって，全く意味のない値が計測されてしまう恐れがある．

5.2 規模と機能による分類

今回の実験では，AuctionClient クラスと AuctionManagerImpl クラスが他のクラスに比べて，動的メトリクスの値が高くなっていた．他のシステムで実験を行ったときに，これらの 2 つのクラスと同程度の規模のクラス，あるいは，同種の機能を持つクラスでの値はどの程度になるのかということを評価する必要がある．

5.3 他の静的メトリクスとの比較

今回の実験では，CBO との比較を行ったが，他にも様々な静的メトリクスが提案されている．他にどのようなメトリクスが提案されているかを挙げていく．

Briand らは，「2 つのクラス c と d に対して，クラス c で実装されているメソッド a がクラス d のメソッド a(新しく定義されている，もしくはオーバーライドされている) を呼び出したり，そのようなメソッドのポインタを受け取るときの関係」である MM(Method-Method interaction) を定義している [2]．また，Lee らは，引数の個数によって重み付けされた他のクラスの呼び出されたメソッドの数で計測される ICP(the Information Flow-Based Coupling) を定義している [8]．更に，Li らは，あるクラスのメソッドで，他のクラスへ送る文の数で定義される MPC(Message Passing Coupling) を定義している [9]．Hitz らは，クラスとオブジェクトの関係の相違を定義している．CLC(Class Level Coupling) は，開発サイクルにおけるクラス間の依存関係と定義し，OLC(Object Level Coupling) は実行時のオブジェクト構造の依存関係である [6]．

このように，様々な静的メトリクスが提案されている．これ以外にも，準備で述べた RFC も静的メトリクスの 1 つである．動的メトリクスとこれらの静的メトリクスを比較して，どのような点で特徴が出て，どのような点で特徴が出にくいかを知る必要がある．静的なメトリクスで計測できないような部分を動的なメトリクスで計測できれば，非常に有用なものとなる．また，静的なメトリクスと比較することによって，今後動的なメトリクスを改良する指標になりうる．この結果は，静的なメトリクスの改良にも有用である．

5.4 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][7]．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 [7] 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．

本論文は，実験の初期結果の報告となっている．今後，実験を引き続き行い，更に詳細な評価をする予定である．

謝辞 本研究は一部，平成 11 年度電気通信普及財団研究調査助成，文部省科学研究費奨励研究 (A)(12780220) の助成を受けている．

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] 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).

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

[7] 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).

[8] 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).

[9] Li, W. and Henry S.: “Object Oriented Metrics that predict Maintainability”, Journal of Systems and Software, Vol. 23, No.2, pp. 111-122 (1993).

[10] 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).