特別研究報告

題目

オブジェクト指向メトリクスを用いた開発支援に関する研究
VC++とMFCを用いた開発を対象として

指導教官
井上克郎教授

報告者
神谷 年洋

平成10年2月12日

大阪大学 基礎工学部 情報工学科

主な用語

メトリクス
オブジェクト指向
フレームワーク
再利用

目 次

1 まえがき

近年，ソフトウェアの応用分野の拡大と共に，ソフトウェアが大規模・複雑化してきている．それに伴い，開発期間の短縮やコストの削減・品質の向上が求められている．これらの要求を実現するために数多くのソフトウェア開発支援に関する研究が行われてきている．

開発支援のアプローチの 1 つはソフトウェア開発における各作業の効率化である．開発作業の効率化を目指してこれまでに多くのソフトウェア開発手法やソフトウェアツールが開発されてきた．最近では，オブジェクト指向パラダイムが注目され，それに基づいた分析法，設計法，プログラミング言語等が数多く提案され，実際の開発現場でも使われるようになってきている [20]．ソフトウェア部品の再利用が効率よく行え，結果として生産性や品質向上が実現できるというのがオブジェクト指向の最も大きな特長となっている．しかし，オブジェクト指向開発プロセスを系統的に管理，支援するための手法は十分に確立されていない．

一方，開発プロセスを改善することで生産性や品質の向上させるという手法も，広く受け入れられている．CMM や ISO9000 は開発プロセス改善のための枠組みとして良く知られている [14][18]．開発プロセスの改善は，通常，(1) 開発プロセスの現状把握と分析，(2) 分析結果に基づく改善策の作成と実行，に分けて実施される．更に，こうした改善を効果的に実施するためには，(1) の現状把握と分析を定量的かつ客観的に行うことが望まれる．そのためには，開発プロセスの状態を表す信頼性の高いデータやメトリクスを用いた分析が必要となる．

ソフトウェアメトリクス [17] は，ソフトウェアプロダクトのさまざまな特性 (複雑度，信頼性，効率など) を判別する客観的な数学的尺度である．メトリクスを用いて開発作業の生産性やプロダクトの状態を評価することで，問題のある作業に対する改善を行う．例えば，エラー分析では，ソフトウェアの開発段階や利用段階で発見されたエラーについて，その原因や混入過程の分析を行ない，その結果に基づいて開発環境を改善することにより，エラーの再発を防止する．

しかし，これらの手法は開発者自身の作業を改善するというよりはむしろ，開発プロジェクト全体を円滑に進めるための改善を目的とする．例えば，ある作業の効率が悪いと判断された場合，その作業に対する開発人員の補充や新しい開発技法の導入といった対策がとられ，開発者自身の作業効率向上を目指すという対策はあまりとられていない．

そこで，本研究ではオブジェクト指向に基づく開発プロセスを対象として，開発者個人の作業効率の向上を目的とした開発プロセスデータの収集とその分析方法について検討する．更に，分析結果を開発者にフィードバックすることの有効性についても議論する．なお，今回の報告では，主に，開発プロセスにおける，コーディングとテスト・デバッグを支援することを目的に開発したプロダクト評価ツールとエラーデータ収集ツールを紹介する。これ

らのツールの応用例として、メトリクスの研究を行った事例を報告する。

2　オブジェクト指向開発

まず、本研究が対象としているオブジェクト指向開発についてここで述べる。

一般に，オブジェクト指向ソフトウェア開発とは，ソフトウェアをいくつかのオブジェクトとその相互作用として開発することである [10]．オブジェクトは開発のすべての工程で一貫して用いることが可能である。例えば、ユースケース法の「アクター」、データベースの設計に用いられるエンティティ-リレーション図 (以下 E-R 図) の「エンティティ」、オブジェクト指向プログラミング言語の「クラス」は、粒度の違いこそあれ、すべてオブジェクトとして捉えられる。

オブジェクト指向開発を支援するための手法，技術として以下のようなものが提案されてきている．実装工程では，オブジェクト指向プログラミング言語が使用されている．さらに，特定のドメインに特化したライブラリであるアプリケーションフレームワークを利用する [10]．アプリケーションフレームワークとは，開発対象となるソフトウェア分野に固有のソフトウェアアーキテクチャを構造に反映したライブラリである．特定の分野に特化することで，大規模な再利用を可能にしている．再利用を行うことで，新規開発部分の規模が小さくなり，開発期間の短縮と品質の向上が可能になる．GUI の分野はフレームワークの実用化がもっとも進んでいて，MFC(Microsoft Foundation Class) 等の商品化されたフレームワークが存在する．分析・設計の工程では，ユースケース法，イベントトレース図，Booch 法 [11] などの手法が提案されてきている．最近では，これら複数の方法を矛盾なく統一的に用いるための UML(UnifiedModeling Language) が提案された [21]．UML という標準的な表記法の登場により，仕様書・設計書の書式が標準化されつつある．

オブジェクト指向開発のための CASE ツールとしては以下のようなものが開発されてきている．

(1) オブジェクト指向仕様作成ツール

(2) コード生成器 (特に視覚的なもの)

(3) コンパイラ，デバッガ，lint に類するツール

(4) テストベンチ，プロファイラ

(5) リバースエンジニアリングおよび報告書作成ツール

(6) レポジトリ，(チーム開発に対応した) バージョンコントロールシステム

さらに，UML が計算機可読文書となり，標準的な文書として用いられるようになれば，(7) オブジェクト指向設計書作成ツールも一般的になると考えられる．

3　計測に基づく開発支援

本研究で対象としているソフトウェア開発プロセスのモデルをここで定式化する。

ソフトウェア開発プロセスとは、特定のリソースを用いて、一連のプロセスを行い、プロダクトを生産することである。

開発プロセスの改善のために、プロセスを測定し、その結果を分析し、その結果に基づいて開発プロセスを変更する手法が良く用いられている。メトリクスは、ソフトウェア開発のプロセスの状況を評価するために必要な尺度である。ソフトウェア開発プロセスの各工程とその成果物 (プロダクト) とその属性、属性を測定するためのメトリクスをまとめたものを図?にあげる (メトリクスの欄は文献 [Fenton1991] から一部引用した)。例えば、このモデルにおいて、プロセスの工程である「要求分析」はプロダクト「要求仕様」を生産する。「要求分析」プロセスについて、かかった「時間」や、「エラーの数」などを測定することができる。このプロセスが生産したプロダクト「仕様」について、その「機能量」や「モジュール性」を測定することができる。機能量を測定するための代表的なメトリクスに「ファンクションポイント (FP)」があり、FP はソースコードの大きさを予測するためなどに用いられる。

メトリクスを用いるために重要なことは、(1) 正確なデータを収集すること、(2) 容易にデータを収集できること、である。正確なデータが収集できなければ、それを用いた分析も不正確になる。また、メトリクスの収集のコストが、分析による生産性の改善を上回ってしまうなら、分析を行う意欲が無くなってしまう (もちろん、ソフトウェア開発プロジェクトが複数実施される場合には一つのプロジェクトで分析するコストを他のプロジェクトに適用することで分散することが可能であり、その場合にはメトリクスの収集と分析のコストがより高い場合も動機付けを行うことができる)。よって、メトリクスを測定する際に一連の計算が必要な、FP や Chidamber らの複雑度メトリクスは、CASE ツールによる支援がの望ましい。

図 プロセス・プロダクト・メトリクス

4　提案する支援環境

本研究で提案する開発支援環境は、プロセスの各工程と、各プロダクトについてメトリクスを用いた計測と分析を行い、開発者にフィードバックすることを目的とする。従来のメトリクスはソフトウェア開発プロセスの管理のために用いられることが多いが、大規模な開発では開発者もメトリクスを用いることが必要となると考えられる。開発者がメトリクスを有効に用いられると考えられる場面は、(1) 既にあるプロダクトを再利用する際にそのプロダクトを評価する、(2) レビューにおいて繰り返しを止める決定を行う、(3) テストの前に、エラーが含まれる部分を予測する、などである。

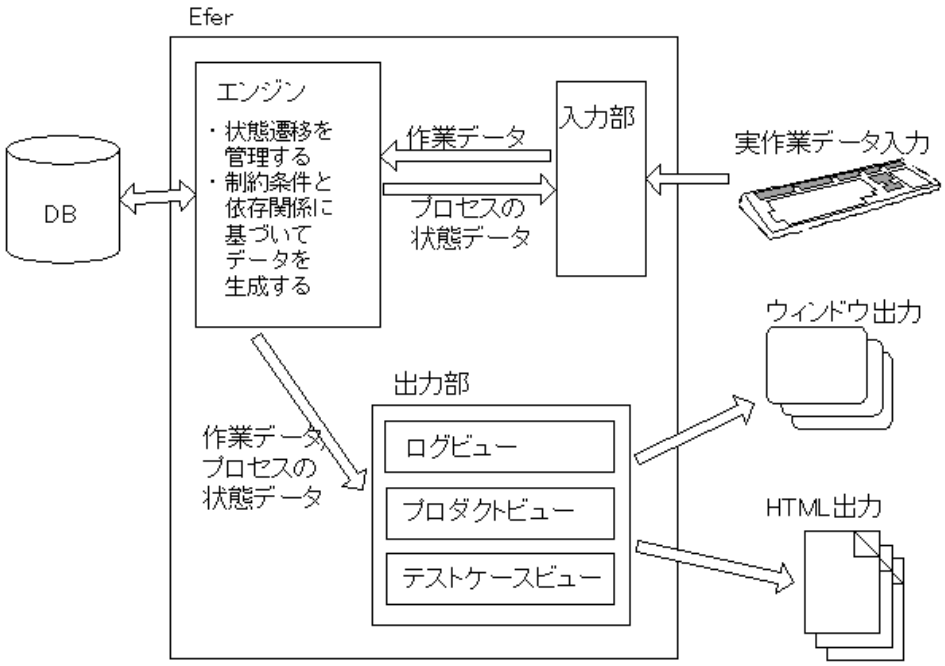

図 1: エラーデータ収集ツール Efer のシステム構成

5 プロトタイプ

プロトタイプとして、提案する開発環境の一部を構成するツールを試作した。試作したツールは、(1) 実装とテストの工程に対してその工数とエラー数を計測するためのツール Efer と、(2) ソースコードに対してその複雑度メトリクスを測定し分析を行うツール Smart である。これらはいずれも Microsoft Windows 95 上で動作し，Visual C++と MFC を用いた開発を対象としている．

5.1 エラーデータ収集ツール Efer

エラーデータ収集ツールは，主に開発者のエラーに関係する作業を記録し，それを分類して表示し，HTML 文書として出力する機能を持つ．プログラムのサイズは C++のソースコードで約 15000 行である．

5.1.1 概要

ツールのシステム構成図を図 5.1.1に示す．ツールは開発作業中つねに実行される．開発者は，ソースコードの変更やレビュー，テストの作業情報をリアルタイムに入力する.「入力

部」はそれらの入力データを，タイムスタンプをつけた上で「エンジン」にわたす．エンジンは入力に基づいてプロセスの状態 (レビュー中，テスト中，フォールトの位置を特定している，など) を遷移させる．状態遷移に基づいて，その時点で開発者が行うことができる作業が変化する．この変化は直ちに入力部に戻される．同時に，記録された作業データとプロセスの状態のデータは出力部に渡される.「出力部」では，それらのデータの書式を整えて，ウィンドウに表示したり，利用者の要求によって HTML ファイルに出力する．

なお，今回試作したエラーデータ収集ツールは，以下の 2 つの作業を対象としてデータを収集する．(1) コードレビューでエラーを発見し，修正する．(2) テストで故障を発見し，その故障の原因であるエラーを特定し，修正する．

5.1.2 収集データ

ツールが自動的に収集するデータは，以下の通りである．

- レビューに要した時間
- エラーの発見に要した時間
- エラーの特定に要した時間
- テストに要した時間
- エラーの修正に要した時間
- エラー ID

一方，ユーザが手作業で入力するデータは以下の通りである．

- 作業内容
- テストケース内容
- エラーの状況
- 修正内容

なお，ツールでは記録を手軽に，正確に行えるようにするため，メニューから選択肢を選ぶことによる入力を多用している (図 2参照)．欠陥やフォールト，クラスは内部のデータベースによって，依存関係が管理されているので，矛盾した記録を行うことは不可能である．たとえば，フォールトが見つかっていなければ，フォールトの修正作業を記録するメニューは現れない．

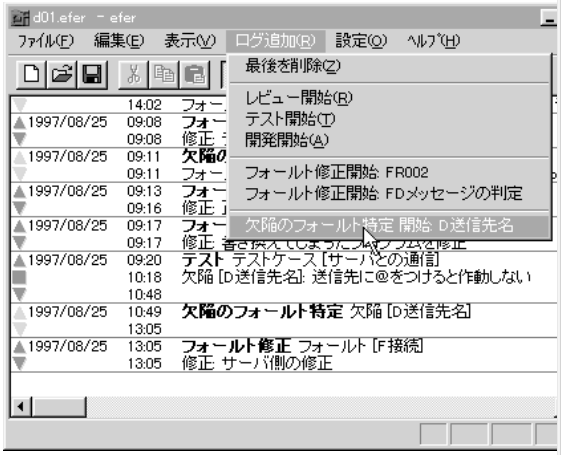

図 2: Efer の記録メニューの例

図 3: Efer 画面例 (A) ログビュー

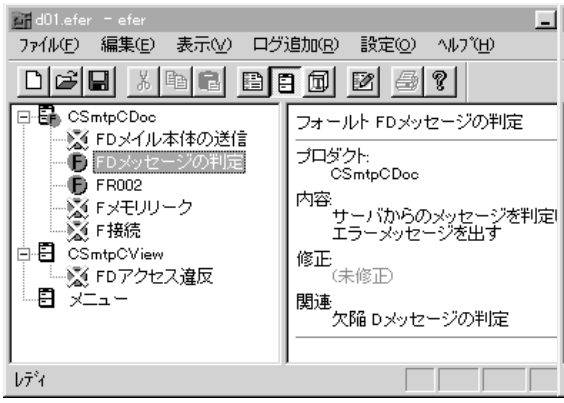

図 4: Efer 画面例 (B) プロダクトビュー

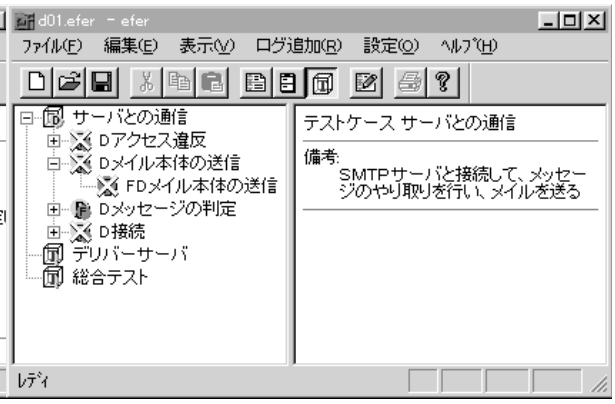

図 5: Efer 画面例 (C) テストケースビュー

5.1.3 画面

ツールは記録を 3 通りの方法で整理して表示する．画面の表示例を示す．同じ内容を，HTML 文書としてファイルに保存する機能も持つ．

(1) ログビュー: すべての作業が時間順に表示される (図 3).

(2) プロダクトビュー：プロダクト (例えばクラス) 単位に，どのようなフォールトがあり，どのように修正されたかが表示される (図 4).

(3) テストケースビュー: テストケース毎に，どのような欠陥が発見されたか，欠陥のフォールトが特定されたかどうか，フォールトが修正されたかどうかが表示される (図 5).

5.1.4 評価

エラーデータ収集ツールを実際にある企業の新人研修プロジェクトに適用し，その使用性について評価 (アンケート) を行った．ツールの利点については，以下のような回答があった．

- ログビューを見ることで，進捗状況を把握しやすかった．
- ログビューに今までの所用時間が記録されているので，作業の見通しが立てやすかった．
- 同じようなエラーがあった場合，今までの記録があるので修正の手助けになった．
- 手作業で記入していたが，ツールを用いることでデータ入力の時間が短縮された．

なお，改善点として，データファイルのバックアップを定期的にとるようにした方がよいというものがあった。

5.2 プロダクト計測ツール Smart

本ツールは複雑度メトリクスを用いて，プロダクトの品質評価を客観的に行うツールである．具体的には，複雑度メトリクスを計測して，プロダクトの中で特に複雑な部分を開発者に通知する機能を持つ．現時点では，標準的な設計書のフォーマットが存在しないため，本ツールはソースコードを対象として分析を行うものとした．

5.2.1 概要

プロダクト計測ツールは C++のソースコードから Chidamber らのメトリクスを計測し，メトリクスに基づいた分析を行うためのツールである．フルセットの C++を対象としている．プログラムのサイズは C++のソースコードで約 1 万行である．ツールのシステム構成

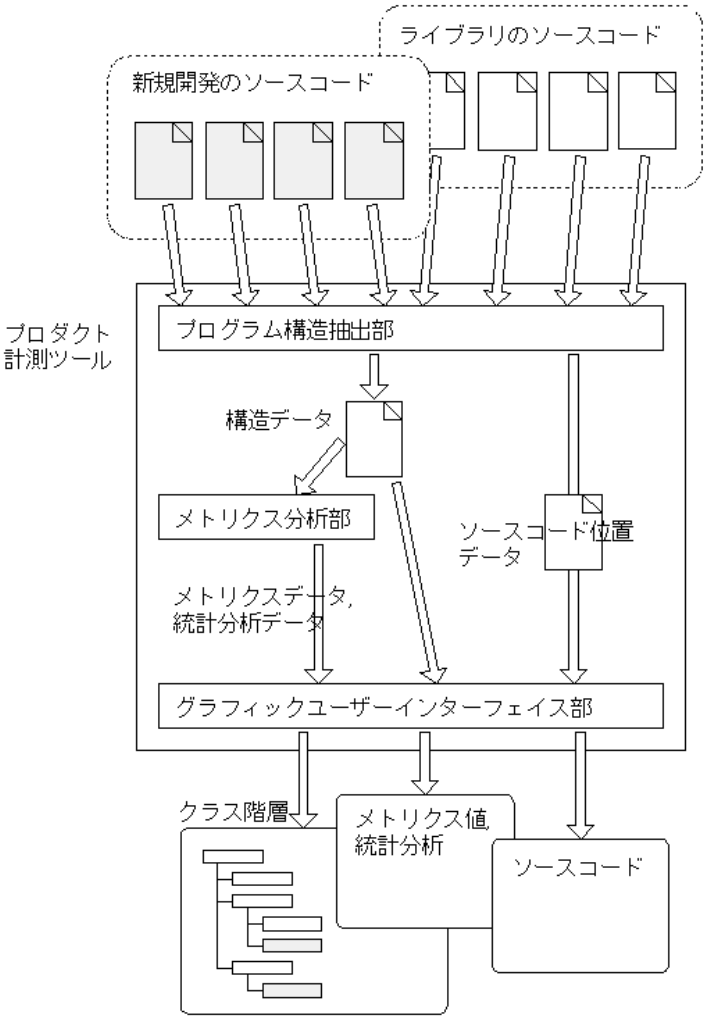

図 6: プロダクト計測ツール Smart のシステム構成

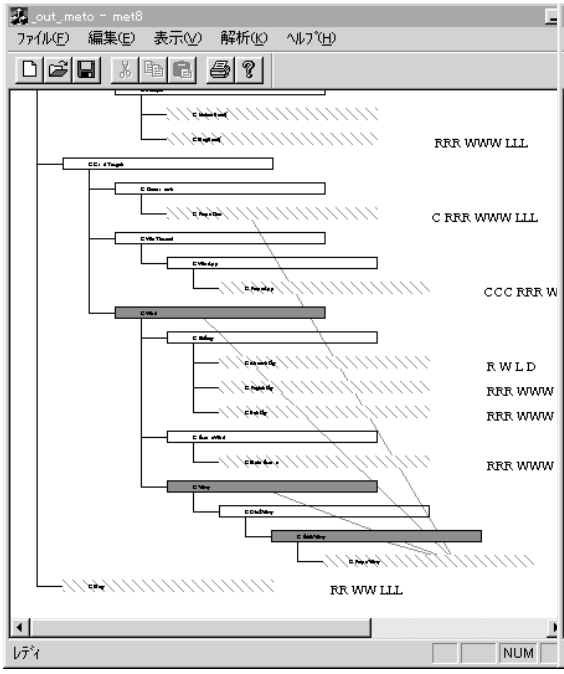

図 7: メトリクスの表示例

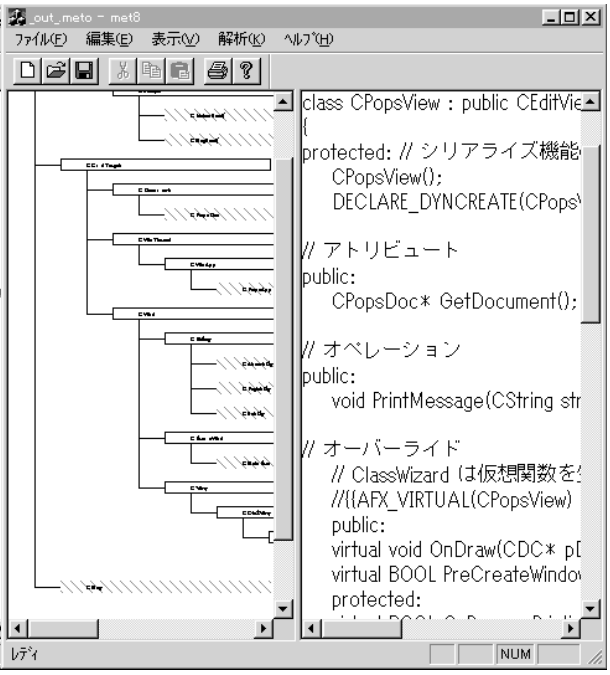

図 8: ソースコードの表示例

図を図 5.2に示す．図中で「プログラム構造抽出部」は，C++ソースプログラムの文法を解析し，定義されているクラスについて，親クラスやインスタンス変数，メソッドなどを識別する．さらに，そのメソッドの定義の内部でどの変数，関数 (あるいは他のクラスのメソッド) を参照しているかを解析する．解析結果を構造データとする．さらに，クラスがソースコード中のどこで定義されているかという情報も，ソースコード位置データとして保存する.「メトリクス分析部」は，構造データをもとに，Chidamber らのメトリクスを計算し，メトリクスの値を統計的に調べることで，異常値を割り出す．これらメトリクスデータ，統計分析データ，構造データ，ソースコード位置データは,「GUI 部」に渡され，利用者の要求によってクラス階層図，メトリクス値や統計分析の結果，ソースコードの表示などが行われる．

5.2.2 機能

ツールが表示する情報は，以下の 3 つである．

(1) **クラス階層図** クラス階層の表示は，(a) フレームワークのクラス階層も含めた全体の階層図，(b) 新規開発の部分と新規開発のクラスが参照するフレームワークのクラス，およびそれらの先祖クラス，の 2 者から選択できる (図 9(a), (b) 参照)．図中の斜線で示される箱が新規開発のクラスである．

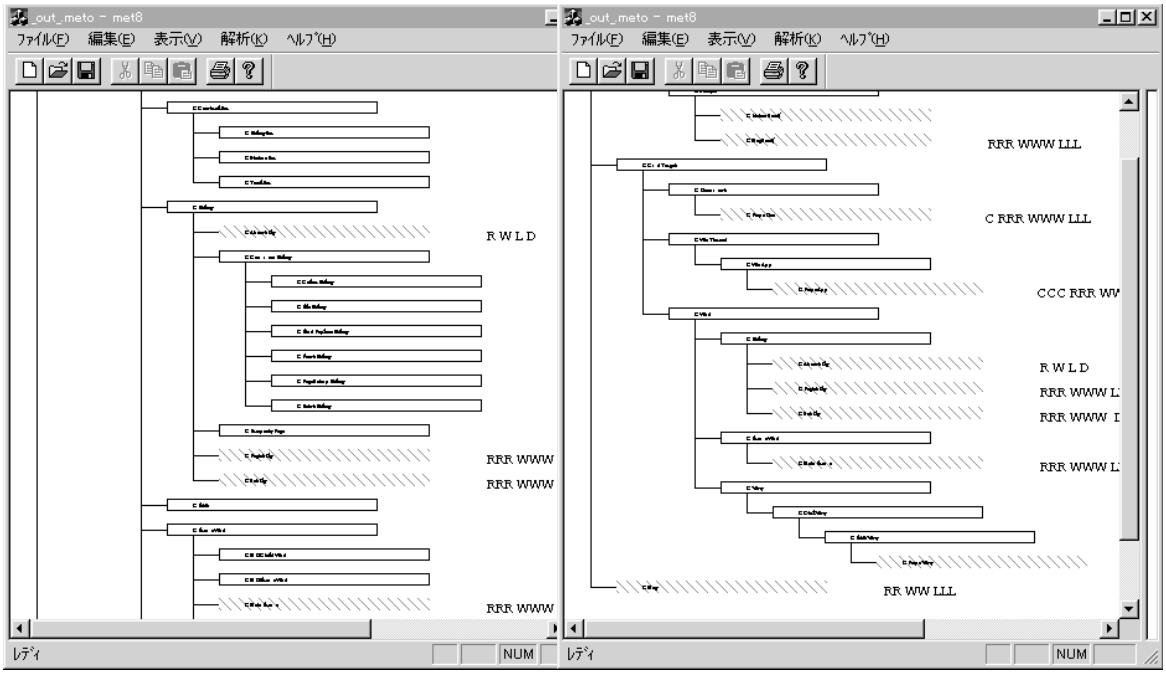

図 9: クラス階層の表示例 (a) 全体 (左), (b) 部分 (右)

(2)Chidamber らのメトリクスの評価結果 クラス階層図上で，選択したクラスに対して Chidamber らの 6 種のメトリクスの評価結果を表示する．特に，クラス間の参照に関するメトリクス RFC と CBO については，結合先のクラスを明示する線を引いて表示することが可能である (図 7参照)．また，基準値を大きく越えている複雑度を持つクラスには，箱の右側にそれを示すマークをつけて示す．例えば,「RW」と表示されていれば，RFC と WMC の値が基準値よりも大きいことを表す.

(3) ソースコード中でクラスを定義している部分 クラス階層図でクラスを指定して，そのクラスを実際に定義しているソースコードを参照することができる (図 8参照).

6　プロトタイプの応用

プロトタイプの応用として、Chidamber らの複雑度メトリクスを修正した実験について述べる。

6.0.3　目的

この実験の目的は，大規模な再利用を行うオブジェクト指向プログラム開発プロセスを対象として，再利用を考慮して Chidamber らのメトリクスを適用する方法を提案し，その評価を行うことである．まず，実際のオブジェクト指向プログラム開発プロセスからデータを収集する．収集したデータに対して，Chidamber らのメトリクスの値と，開発プロセス中でエラーを修正するために要した時間との関係を調べる．次に，オブジェクトクラスの再利用を考慮した Chidamber らのメトリクスの適用方法を述べ，この適用方法により予測性が改善されることを示す.

6.0.4　Chidamber らの複雑度メトリクス

Chidamber と Kemerer が提案した 6 つのメトリクス [2] は，オブジェクト指向設計のための複雑度メトリクスであり，クラスの定義からその複雑度を測定する．いずれも数値が大きいほど，より複雑であることになる．

WMC Weighted Method per Class; クラス当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程度の複雑度であると仮定できるときは，評価対象の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るときは，評価対象のクラスから根に到るまでのパスの長さ．

NOC Number Of Children; 子クラスの数 評価対象のクラスから直接派生しているクラスの数である．

CBO Coupling Between Object class; クラス間の結合 評価対象のクラスが “結合” しているクラスの数である．結合とは，あるクラスが他のクラスの属性やメソッドを参照することを意味する．

RFC Response For a Class; クラスに対する反応 評価対象のクラスのメソッドの集合と，そのクラスのメソッドが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である．

LCOM Lack of Cohesion in Methods; メソッドの凝集の欠如 評価対象の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が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である．

Chidamber らは 2 つのソフトウェア開発組織でオブジェクト指向言語 (C++と Smalltalk) を用いて開発されたプログラムに含まれるクラスからこれらのメトリクスの値を算出し，クラス毎のメトリクスの値の平均値が大きいほど開発費用が大きくなることを実験的に確かめている [2]．しかし，メトリクスを計測する際，再利用されるクラスは新規開発されるクラスよりも品質が高いことは考慮されていない．

6.0.5 メトリクスの適用方法

Chidamber らのメトリクスは，再利用されるクラスと新規開発されるクラスを区別せずに扱っている [2]．しかし，再利用されるクラスは新規開発のクラスよりもエラーが少ない (これらは Basilli ら [1] によっても観察されている) ので，新規開発のクラスよりも複雑度が低いと思われる．そこで，次のような仮説を立てる：Chidamber らのメトリクスにおいて，再利用されるクラスに対する「結合」や「参照」は複雑度を増大させない．これに従い，Chidamber らの 6 種のメトリクスの定義を，再利用されるクラスの重みを 0 とするように修正する (新規開発のクラスの重みは従来通り 1 とする)．**??**で述べたメトリクスの中で，他のクラスとの結合や参照を評価するものは DIT, CBO, RFC である．残り 3 つの WMC, NOC, LCOM は一つのクラス内で閉じたメトリクスであるため値は変化しない．ここで，修正したメトリクスをそれぞれ DITo, CBOo, RFCo とし，次のように定義する．

DITo 継承木の深さ クラスの派生関係が木であると仮定できるときは，評価対象のクラスから根に到るパスの長さから，パス上にある再利用されたクラスの数を引いた数である．

CBOo クラス間の結合 評価対象のクラスが結合しているクラスのうち，開発者が開発したクラスに対する結合の数である．

RFCo クラスに対する反応 評価対象のクラスのメソッドの集合と，評価対象のクラスのメソッドが呼び出す他のクラスのメソッドのうち開発者が開発したクラスに属するメソッドの集合との和集合の要素数である．

6.0.6 実験概要

日本ユニシス株式会社の 1996 年度新人研修における C++プログラム開発演習からデータを収集した．研修生 (被験者) は演習の前に，オブジェクト指向設計，オブジェクト指向言

語について講習を受けている．この演習では，6 つのチームが独立に同じ課題を行った．各チームは 4〜5 名の被験者で構成されている．

開発プロセスはウオーターフォールモデルで行われた．すなわち，要求仕様定義，設計，コーディング，レビュー，単体テスト，結合テストのフェーズを経た．課題プログラムはいわゆる酒屋問題 [8] を拡張したもので，データベースを用いた在庫管理，パスワードによるオペレータ認証，売り上げデータのグラフィカルな表示，売上予測等の機能を持つ．課題が渡された時点で，データベースの構造，入出力ファイルフォーマット，および被験者が開発すべきサブシステム (パスワード管理サブシステム，等) が決定されている．つまり，要求仕様定義フェーズと設計フェーズの一部が終了していることになる．開発期間は 5 日間である．開発されたプログラムは，インストラクタによってテストされ，要求仕様を満たすことが確認される．

プログラミング言語は C++であり，処理系は Microsoft Visual C++である．フレームワークとして Microsoft Foundation Class(MFC) を用いた．MFC を用いることは課題の重要な要件であり，ユーザーインターフェイスとデータベースインターフェイスはすべて MFC のクラスを用いて実装される [9]．図 10に，本実験において，あるチームによって開発されたアプリケーションのクラス階層を示す．図 10では，新規開発されたクラスは網掛けで示されている．新規開発のクラスはすべてクラスライブラリのクラスから派生していることがわかる．

今回の開発では大規模な再利用が行われている．新規開発部分については，行数でチーム当たり 3000 行程度であり，これには空白行やツールによって生成された行が含まれる．また，開発されたクラスは，すべてクラスライブラリから派生したものである．一方，再利用した部分については，行数でチーム当たり 10000 行程度である。

6.0.7 分析

開発はチームを構成して行われているが，課題プログラムは独立した部分プログラムに分割され，チームのメンバーに割り当てられる．実際に，部分プログラム間に渡るようなエラーはほとんど発見されておらず，各開発者はチームの他のメンバーの開発による影響を受けていない．従って，以降の分析は被験者単位で行っている．

なお，収集されたデータに不備のあった被験者は分析の対象から除いた．結果的には，19 人のデータが分析対象となった．

表 1に，各被験者の開発したプログラムのコードレビュー直前のソースコードについて調べた各メトリクスの値を示す．メトリクスの値は各被験者が開発したすべてのクラスについての合計値である．ただし，NOC についてはすべての被験者が開発したプログラムにつ

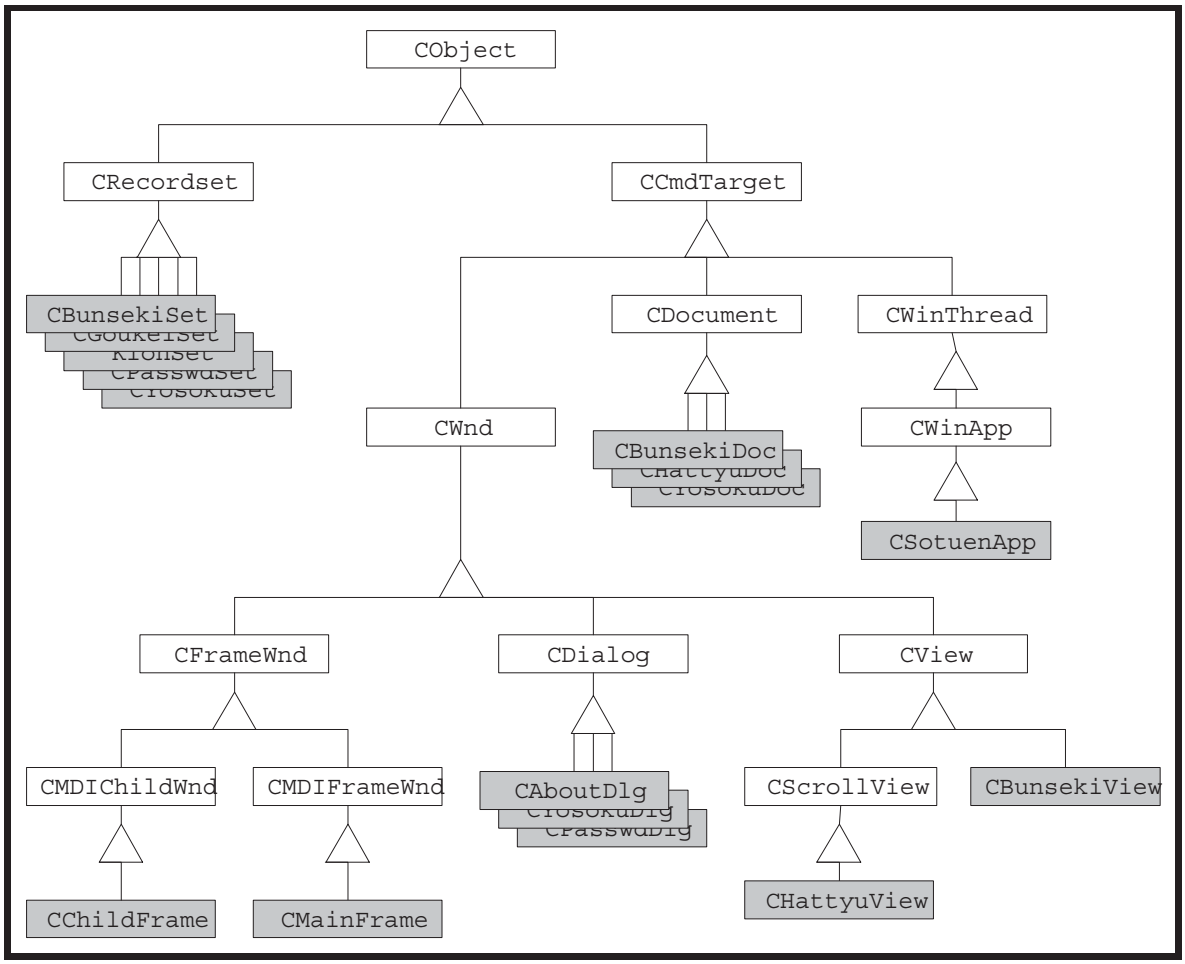

図 10: クラス階層の例

表 1: 6 種のメトリクスとエラーの相関

	WMC	CBO(CBOo)	RFC(RFCo)	LCOM	DIT
Ec	0.62	0.58	0.54	0.65	0.68
Et	0.72	0.74(0.0.47)	0.63(0.77)	0.7	0.77

いて，値が 0 であったため省略している．

CBOo と Et の相関は 0.47 であった．一方，CBO と Et の相関は 0.74 であった．CBOo は CBO よりも予測性が低下している．これは，CBO については，フレームワークからのクラスと新規開発のクラスが同じくらい複雑度に寄与していることを意味する．この理由は，相手のクラスの属性を直接参照するような “結合” を行う場合，結合しているクラスの定義を理解していなければならないため，結合しているクラスが再利用されたクラスであっても，開発者が新規に開発したクラスと同様に，複雑度を増大させるからであると考えられる．

RFCo と Et の相関は 0.77 であった．一方，RFC と Et の相関は 0.63 であった．RFCo は RFC よりも予測性が向上している．これは，RFC については，フレームワークのクラスは新規開発のクラスほど複雑度に寄与していないことを意味する．

結果として，大規模な再利用が行われている場合では，CBO については，再利用された部分も対象にして評価を行う方が，より正確に複雑度を測定できる．一方，RFC については，再利用された部分は対象とせずに評価を行う方がより正確に複雑度を評価できることが確認された．

参考文献

[1] V. R. Basili, L. C. Briand, and W. 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)

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

[3] 本位田真一, 青山幹雄, 深澤良彰, 中谷多哉子:“オブジェクト指向分析, 設計”, 共立出版 (1995)

[4] S. H. Kan: “Metrics and Models in Software Quality Engineering”, Addison-Wesley (1995)

[5] 久米均, 飯塚悦功：“回帰分析”，岩波書店 (1987)

[6] 松本健一, 井上克郎, 菊野亨, 鳥居宏次：“エラー寿命に基づくプログラマ性能の実験的評価 -大学環境におけるプログラム開発-”, 電子情報通信学会論文誌 D, Vol.J71-D, No.10, pp.1959-1965 (1988-10)

[7] 山田茂, 高橋宗雄：“ソフトウェアマネジメント入門”，共立出版 (1993)

[8] 山崎利治：“共通問題によるプログラム設計技法解説”，情報処理学会誌，25，9，p. 934 (1984)

[9] “Visual C++ブック”，マイクロソフト (1996)

[10] 青木淳: “オブジェクト指向システム分析設計入門”, 株式会社ソフト・リサーチ・センター (1993).

[11] G. Booch:“Object-Oriented Analysis and Design with Applications, 2nd Edition”, The Benjamin/Cummings Publishing Co., Inc(1994).

[12] C. R. Cymons:“Function Point Analysis: Difficulties and Improvements”, IEEE Transaction on Software Engineering, Vol. 14, No. 1, pp.2-10(1998).

[13] IFPUG: “Function Point Counting Practices Manual, Release 4.0”, International Function Point Users Group(1994).

[14] 飯塚悦功 編:“ソフトウェアの品質保証 ISO-9000-3 対訳と解説”, 日本規格協会 (1992).

[15] 神谷, 別府, 楠本, 井上, 毛利:“オブジェクト指向プログラムを対象とした複雑度メトリクスの実験適評価”, 電気学会論文誌 C 117 巻 11 号, pp. 1586-1592(1997).

[16] M. Lorenz and J. Kidd:“Object-Oriented Software Metrics — A Practical Guide”, PTR Prentice Hall, Inc.(1994). 宇治邦明 監訳, オージス総研 訳:“オブジェクト指向ソフトウェアメトリス — 現実的な運用のためのガイド:”, 株式会社トッパン (1995).

[17] P. Oman and S. L. Pfleeger:“Applying Software Metrics”, IEEE Computer Society Press(1997).

[18] M. C. Paulk, et al:“The Capability Maturity Model: Guidelines for Improving the Software Process”, Addison Wesley Publishing Co., Inc.(1995).

[19] M. Pighin and R. Zamolo:“A Predictive Metric Based On Discriminant Statistical Analysis”, Proceeding of 19th ICSE, Boston, Massachusetts, USA, pp. 262-270(1997).

[20] D. Takach and R. Puttick:“Object-Technology in Applications Development”, Addison Wesley Publishing Co., Inc.(1994)., 上野南海雄, 守屋政美 監訳:“アプリケーション開発のオブジェクト指向テクノロジー”, アジソン・ウェスレイ・パブリッシャーズ・ジャパン (1997).

[21] Rational Software Co.: “UML Summary”, ver. 1.1(1997). (taken from http://www.rational.com/)

[22] 山田茂, 高橋宗雄:“ソフトウェアマネジメントモデル入門 — ソフトウェアの品質の可視化と評価法”, 共立出版株式会社 (1993).