C++プログラムを対象とした複雑度メトリクス計測ツールとその評価

神谷 年洋　高林 修司　楠本 真二　井上 克郎[†]

ソフトウェアの品質や生産性の向上を目的として，開発支援のために数多くの手法が提案され，実践されている．代表的なものとしては，各種CASEツールやソフトウェアメトリクス，プロセス成熟度モデルといったものがある．しかし，これらの技術による開発支援は，主に管理のために用いられており，開発者に対する支援として用いられることは少ない．従って，開発現場へのこれらの技術の導入は困難なものになっている．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した．具体的には，今回開発した，C++のソースコードを対象としたプロダクト評価ツールの概要と使用例について述べる．

Measurement tool for complexity of C++ program

Toshihiro Kamiya　Shuji Takabayasi
Shinji Kusumoto　Katuro Inoue[†]

For the purpose of improving quality and productivity of software, many methodologies have introduced in the software development process. For example, object-oriented technologies, CASE tools, software metrics, and process maturity models are representative ones. However, they are primarily used for supporting process management rather than supporting developing activity. Therefore, it is difficult to introduce these technologies into development process. In this paper, we describe a method for collecting and analyzing process data of object-oriented software development, aiming for software engineer's personal skill improvement. We have developed a tool as developing environment, by which software engineers collect metrics values from C++ source code and analyze them statistically.

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[13)]．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．

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[7)11)]．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[10)]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．

しかし，これらの手法は開発者自身の作業を改善す

[†] 大阪大学 大学院基礎工学研究科
Division of Software Science, Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University

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．

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について紹介する．

2. ソフトウェアメトリクス

2.1 メトリクスとその利用

ソフトウェアメトリクスは，ソフトウェアのさまざまな特性 (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り，ソフトウェアを統計的な視点から見ることを可能にする[9)]．

例えば，分析フェーズでは，仕様書からソフトウェアの機能の大きさを測定し，それによって開発コストを見積もるためのFP(ファンクションポイント) が提案されている[5)]．設計や実装のフェーズでは，設計書やソースコードからソフトウェアの複雑さを測定し，それによってエラーの数を予測するための複雑度メトリクスがよく用いられている[2)12)]．従来の (オブジェクト指向ではない) ソフトウェアに対しては，FP としてIFPUG 法[6)]，複雑度メトリクスとしてHalsted のメトリクス，McCabe のサイクロマチック数[15)]等がある．また，オブジェクト指向ソフトウェアに対する複雑度メトリクスには，Chidamber らのメトリクス[4)] が有名である．

前書きでも述べたように，プロセス改善において，これらのメトリクスは主に管理のために用いられており，開発者に対する支援として用いられることは少ない．開発者が複雑度メトリクスを用いることで，プログラム中で特に複雑になっている部分を特定することができ，プログラムの構造設計などの修正を行うかどうかの判断や，テストを重点的に行う場所の特定が可能になる．特に，開発が個人ではなくチームで行われる場合や，別の開発者が開発したソースコードを引き継いだ場合など，開発者が細部を知らないプロダクトを扱わなければならないときには，より客観的に評価できること，より抽象化された (要約された) 情報で評価できること，は重要な利点になる．

2.2 Chidamber らの 6 種のメトリクス

オブジェクト指向ソフトウェアに対する複雑度メトリクスとして，Chidamber らは 6 種類の複雑度メトリクスを提案している[4)]．

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クラスに含まれるメソッドの集合の和集合をとり，その要素数をとる．

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[4)]．また，これらのメトリクスは，オブジェクト指向で開発されたソフトウェアに対して，従来の複雑さメトリクスよりもエラーの個数と相関が高いことが示されている[2)]．

我々も，フレームワークを用いた開発を対象として，クラスの再利用が CBO と RFC に影響を与えることを明らかにし，その適用方法について検討している[8)]．

3. オブジェクト指向開発支援

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[1)]．オブジェクトは開発

の分析・設計・実装のフェーズまで一貫して用いることが可能である．

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．

実装フェーズでは，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[1]．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・設計のフェーズでは，ユースケース法，イベントトレース図，Booch 法[3]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(Unified Modeling Language) が提案された[14]．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．

しかし，これらのツールにおいて，メトリクスを用いて統計的な視点からプロダクトを評価するもの (以下メトリクスツール) は，ほとんど存在しない．メトリクスツールは，開発コストの見積もり，設計の評価・洗練，テストスケジュールの決定，プロダクトの品質評価等の目的に用いることができる．さらに，特にチーム開発において，客観的な評価基準があることは，設計における意思決定や，レビューに有効である．

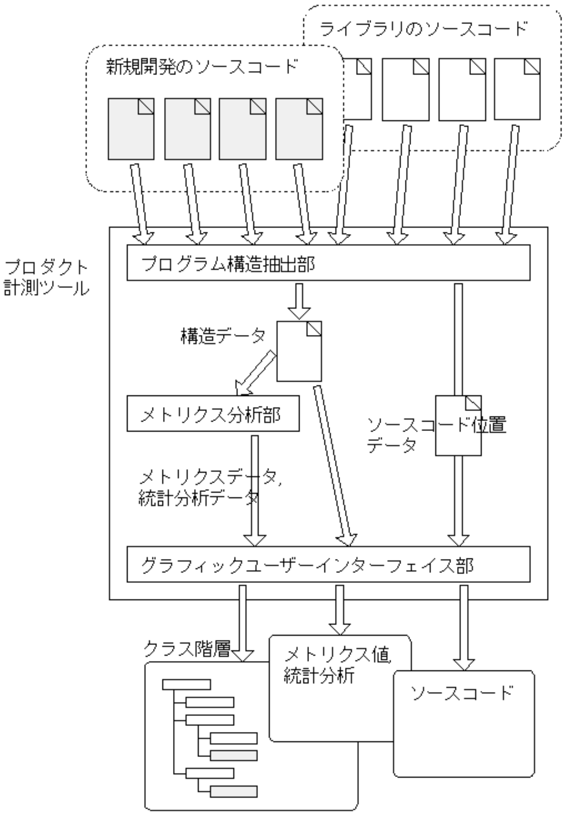


図 1 プロダクト計測ツールのシステム構成
Fig. 1 System structure of product metrics tool

4. プロダクト計測ツール

4.1 基本方針

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定量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．

4.2 ツールの概要

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を図 1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．

4.3　ツールの機能と特長

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 2(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．

(2)Chidamber らのメトリクスによる複雑度評価

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 3参照）．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（詳細は次節）．

(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 4参照）．

なお，「プログラム構造抽出部」が独立しているため，この部分を取り替えることで，C++以外のオブジェクト指向プログラミング言語への対応が可能である．また，メトリクス分析部を変更することで，他のメトリクスの追加も容易に行える．

4.4　メトリクスによる複雑度判定の方針

本ツールが採用したメトリクスは，いずれも測定値が大きいほど複雑であることを意味する．メトリクスの値が大きなクラスを見つけ出せば，それが複雑なクラスとなる．また，クラスの種類（例えば，ユーザーインターフェイスを受け持つクラスか，データベースにアクセスするクラスか）によって，メトリクス値の分布や有効性に大きな違いがあることが指摘されている[2]．そのため，本ツールはクラスの種類ごとに基準値を設定し，計測対象のクラスの測定値と基準値との差からそのクラスが複雑であるかどうかを判定する．

4.4.1　クラスの分類

クラスを機能別に分類する．具体的には，あるクラスの親（または先祖）がフレームワークのどのクラスであるかによって，クラスを分類した．フレームワークのクラス階層がほぼ機能別に部分木に別れているため，親クラスを見ることで，機能によるクラスの分類が可能になる．ただし，この分類方法はフレームワークやアプリケーションドメインに依存する．

4.4.2　メトリクスの基準値の設定

クラスの分類ごとに，過去に収集されたクラスのメトリクス値から，メトリクスの平均値と標準偏差を求める．

4.4.3　異常値の算定

あるクラスのメトリクスの測定値を v, そのクラスが属する分類におけるメトリクスの平均値を m, 標準偏差をσとすると，メトリクスの異常値 i は，次式で計算される．

$$i = min(max(0, \lfloor (v-m)/\sigma \rfloor), 3)$$

$i > 0$ の場合は複雑であると判断し，クラス名の後ろにメトリクスの頭文字を i 文字並べることで通知する（図 3参照）．

5. 評価実験

5.1　概　　要

本ツールを，1997 年度の日本ユニシス株式会社で行われた新人研修での開発演習で得られたデータに適用した．演習課題は，電子メールの配送システムの作成であり，システムは 5 つのサブシステム（SMTP サーバ，POP サーバ，DELIVER サーバ，SMTP クライアント，POP クライアント）から構成されている．開発チームは 4 から 5 人の開発者で構成され，開発者は各サブシステムを開発する．演習開始時に開発チームに各サブシステムの仕様が渡され，6 日間で，設計，実装，テストを行う．インストラクターによる受け入れテストに合格した時点で開発が終了する．プログラミング言語として Visual C++を用い，フレームワークには MFC を用いた．開発規模はチームあたり 3000 行程度（再利用分を含まない）である．

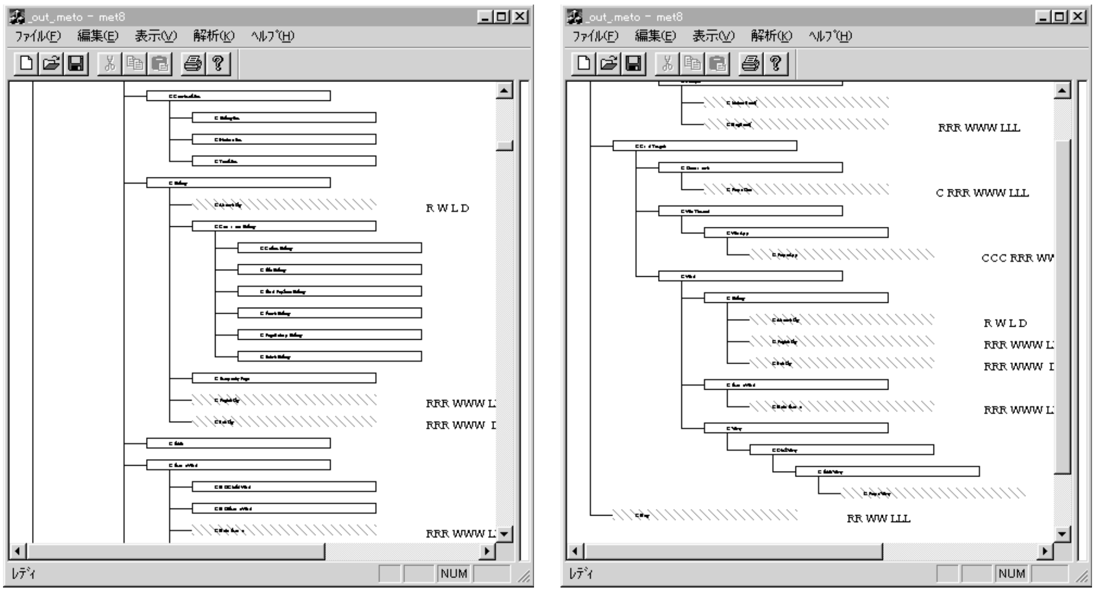


図 2　クラス階層の表示例 (a) 全体 (左), (b) 部分 (右)
Fig. 2　Sample of class tree (a)All(left), (b)Partial(right)

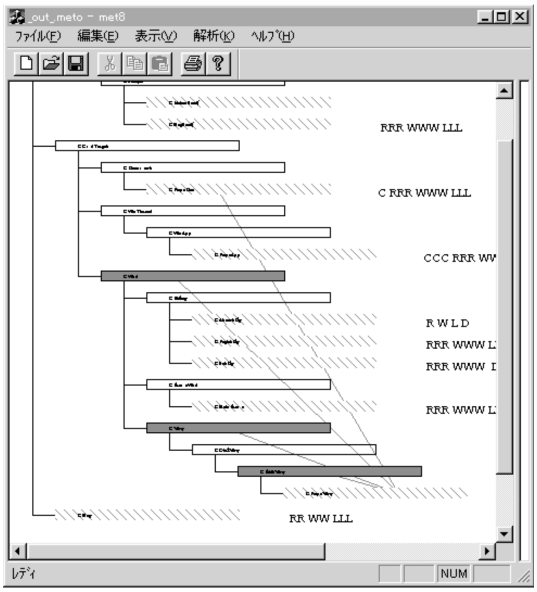
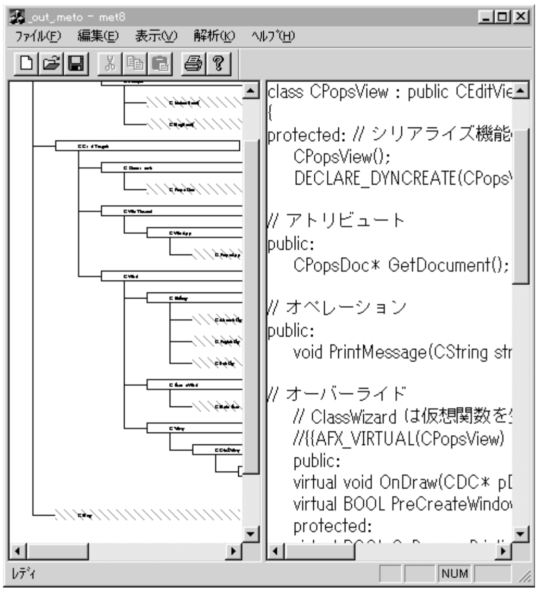


図 3　メトリクスの表示例
Fig. 3　Displaying metrics values



```
class CPopsView : public CEditVie
{
protected: // シリアライズ機能
    CPopsView();
    DECLARE_DYNCREATE(CPops

// アトリビュート
public:
    CPopsDoc* GetDocument();

// オペレーション
public:
    void PrintMessage(CString str

// オーバーライド
    // ClassWizard は仮想関数を
    //{{AFX_VIRTUAL(CPopsView)
    public:
    virtual void OnDraw(CDC* pD
    virtual BOOL PreCreateWindo
    protected:
```

図 4　ソースコードの表示例
Fig. 4　Displaying source code

5.2　クラスの分類

クラスの分類は以下の 6 種類とした．

(1)**CDocument** 派生クラス（ドキュメントクラス）　親クラスが CDocument であるクラスは，プログラムのデータを処理する部分が記述される．

(2)**CView** 派生クラス（ビュークラス）　親クラスが CView であるクラスは，ユーザーに対してデータを表示する部分が主に記述される．

(3)**CDialog** 派生クラス　親クラスが CDialog であるクラスは，ユーザーからデータを受け取る部分と，ユーザーに対してエラーメッセージを出す部分が主に記述される．

(4)**CWinApp** 派生クラス　親クラスが CWinApp であるクラスは，Windows アプリケーションに

特有の振る舞い(「アプリケーションの前回のウィンドウの位置と大きさを覚えておく」など)が記述される．

(5)CFrameWnd 派生クラス　複数のビューを持つプログラムの場合，それらを管理するためのコードが CFrameWnd から派生したクラスに記述される．(ユーザーインターフェイスが複雑になると，複数のビューを切り替える方法は良く用いられる．)

(6) その他　(1)～(5) のいずれにも該当しないクラス．親クラスが無いクラスも含む．

5.3 実験データ

開発された 17 人分のプログラムから抽出したメトリクスデータを図 5から図 10に示す．この図では，クラス分類ごとに，別々のグラフにメトリクス値をプロットしている．各グラフの，細い線で描かれた一つの多角形が，一つのクラスに付いてのメトリクス値を表わす (メトリクスの値は，すべてのクラスについての平均が 1.0 となるように正規化されている)．太い線で描かれた多角形は，その分類に属するすべてのクラスのメトリクスの平均値である．CBO, RFC, WMC, LCOM, DIT は Chidamber らの 6 種のメトリクス (NOC は 0 であるため省略)，NIV はクラスのインスタンス変数の数，SLOC はクラスのソースコードの行数である．

5.4 分析

図 5から図 10のグラフから，その他を除く 5 つの分類について，分類ごとにメトリクスの分布のパターンがあることが分かる．つまり，分類ごとに異常値の判断基準を変えることで，より精度の高い分析が可能であると考えられる．また，分類 CWinApp に関しては，すべてのメトリクスについてメトリクス値の分散がほぼ 0 であり，メトリクスによる複雑度分析が上手く行かないことが分かる．

これらのメトリクスデータから，クラスの異常値を計算した．ある開発者が開発したプログラムについて，メトリクスの異常値と，発見されたエラーの個数および修正に要した時間を表 1に示す．i_{CBO}，i_{RFC}，…，i_{SLOC}はそれぞれのメトリクスの異常値，i_{sum}は異常値の合計，Ec はそのクラスで修正されたエラーの数，Et は修正に要した時間 (単位 :分) である．異常値が高いクラスにエラーが集中していることが分かる．

17 人分のすべてのクラスについて，異常値と Ec，異常値と Et について，Spearman の順位相関を求めたものをそれぞれ表 2，表 3に示す (同順位補正を行っている)．i_{sum}と Ec および Et に順位相関関係があることが，有意水準 1 %で確認された．これにより，ツールが複雑であると指摘するクラスについて，実際にエラーが発生していることが確認された．

6. まとめ

本研究では，オブジェクト指向に基づくソフトウェア開発プロセスにおける，コーディングとデバッグを支援することを目的としたプロダクト評価ツールについて述べた．現在，ツールの有効性を評価するために収集したデータのさらなる分析を予定している．

今後の課題としては，次の 3 点があげられる．

(1) より多くのプロジェクトに対して，ツールの有用性を評価する．

(2) プロダクト評価ツールにおいて，Chidamber らのメトリクス以外のメトリクスを適用できるようにする．

(3) 上流工程の CASE ツールのデータを扱えるようにする．

謝辞

評価実験に御協力頂いた日本ユニシス株式会社の毛利幸雄氏に深謝致します．

参考文献

1) 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).

2) V. R. Basilli, L. C. Briand, and W. L. Melo: “A Validation of Object-Oriented Design Metrics as Quality Indicators”, IEEE Transaction on Software Engineering, Vol.20, No.22, pp.751-761(1996).

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

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

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

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

7) 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).

8) 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

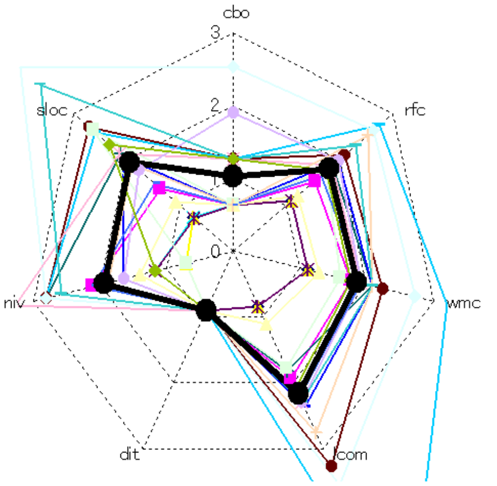


図 5 CDocument 派生クラス (サンプル数 19)
Fig. 5 Classes derived from CDocument(19 samples)

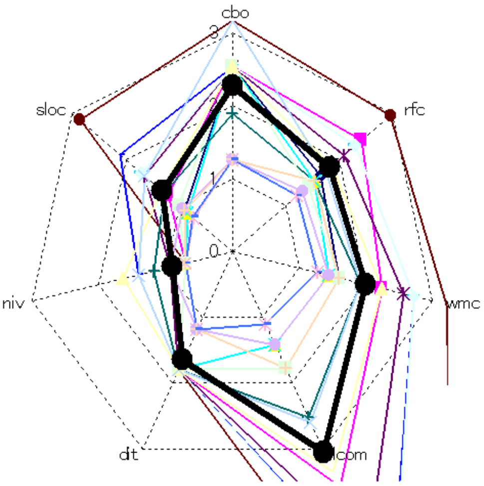


図 6 CView 派生クラス (サンプル数 17)
Fig. 6 Classes derived from CView(17 samples)

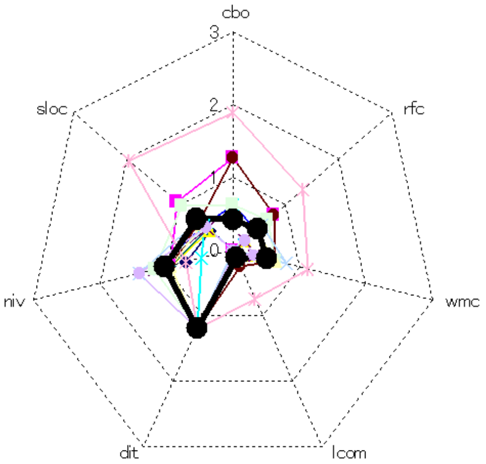


図 7 CDialog 派生クラス (サンプル数 15)
Fig. 7 Classes derived from CDialog(15 samples)

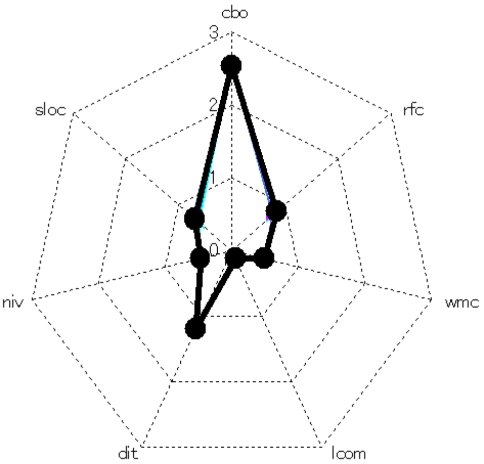


図 8 CWinApp 派生クラス (サンプル数 17)
Fig. 8 Classes derived from CWinApp(17 sampels)

験的評価”, 電気学会論文誌 C 117 巻 11 号, pp. 1586-1592(1997).

9) 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).

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

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

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

13) D. Takach and R. Puttick:“Object-Technology in Applications Development”, Addison Wesley Publishing Co., Inc.(1994)., 上野南海雄, 守屋政

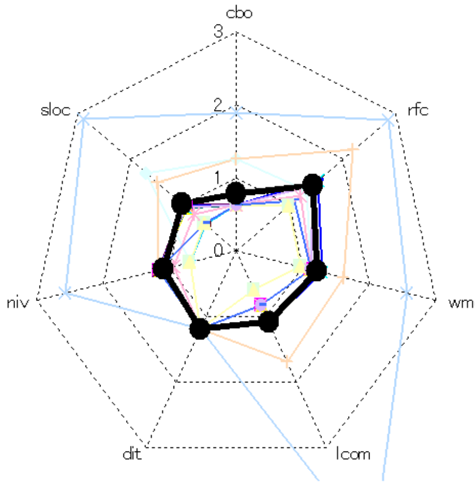


図 9 CFrameWnd 派生クラス (サンプル数 17)
Fig. 9 Classes derived from CFrameWnd(17 samples)

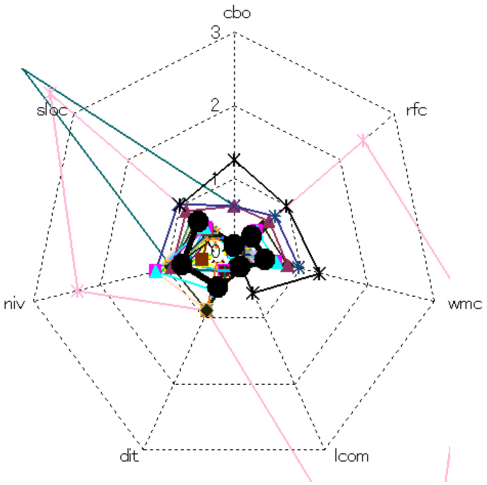


図 10 その他のクラス (サンプル数 39)
Fig. 10 Other classes(39 samples)

表 1 あるプログラムについてのクラスの異常値とエラー
Table 1 Illegalness value and error of a program

クラス	i_{sum}	i_{CBO}	i_{RFC}	i_{WMC}	i_{LCOM}	i_{NIV}	i_{SLOC}	Ec	Et
CMailFile	9	3	2	2	1	0	1	1	77.9
CIndex	6	1	2	1	0	1	1	0	0
CPopServerDoc	5	2	2	0	0	0	1	1	0.1
CPasword	1	0	1	0	0	0	0	1	41.4
CConfig	1	0	1	0	0	0	0	1	8.3
CPopServerView	0	0	0	0	0	0	0	0	0
CPopServerApps	0	0	0	0	0	0	0	0	0
CMessageSock	0	0	0	0	0	0	0	1	0.3

表 2 異常値と Et の順位相関係数
Table 2 Rank correlation between illegalness value and Et

	i_{sum}	i_{CBO}	i_{RFC}	i_{WMC}	i_{LCOM}	i_{NIV}	i_{SLOC}
相関係数	0.536	0.457	0.495	0.348	0.472	0.334	0.525

表 3 異常値と Ec の順位相関係数
Table 3 Rank correlation between illegalness value and Ec

	i_{sum}	i_{CBO}	i_{RFC}	i_{WMC}	i_{LCOM}	i_{NIV}	i_{SLOC}
相関係数	0.538	0.451	0.489	0.341	0.484	0.343	0.513

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).
14) Rational Software Co.: “UML Summary”, ver. 1.1(1997). (taken from http://www.rational.com/)
15) 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).