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新人研修卒業演習への適用

神谷 年洋†, 楠本 真二†, 井上 克郎†, 毛利 幸雄‡

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, 開発現場へこれらの技術を導入すること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新人研修卒業演習での適用例について述べる.

Complexity metrics tool for C++ and its case study

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

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. Then we applied the tool to the data collected from the software development project of newcomer training course.

1. はじめに

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[14]. 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[6] [10]. 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

†大阪大学大学院基礎工学研究科
Graduate School of Engineering Science Osaka University
‡日本ユニシス株式会社

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[9]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. また, 新人研修卒業演習により得られたデータをツールによって分析し, 卒業演習の結果についての考察を行った.

2. 準備

2.1. オブジェクト指向複雑度メトリクス

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[8].

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[4]. 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] [11]. 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[5], 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](以下 C&K メトリクス)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.

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(C&K メトリクス)を提案している[4].

WMC(クラスの重み付きメソッド数;Weighted Methods per Class):

計測対象クラス C_1 が, メソッド $M_1, \ldots, M_n$ を持つとする. これらのメソッドの複雑さをそれぞれ $c_1, \ldots, c_n$ とする. このとき, $WMC = \Sigma c_i$ である. 適切な間隔尺度 f を選択して $c_i = f(M_i)$ によりメソッドを重み付けする.

文献[2]と[4]においては, すべてのメソッドの複雑さが同じであるという仮定をおいて, WMC をメソッドの数とした. この論文でも同じ仮定を用いる.

DIT(継承木における深さ;Depth of Inheritance Tree):

DIT は計測対象クラスの継承の深さである. 多重継承が許される場合は, DIT は継承木におけるそのクラスを表す節点から根に至る最長パスの長さとなる.

NOC(子クラスの数;Number Of Children):

NOC は計測対象クラスから直接導出されているサブクラスの数である.

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

CBO は, 計測対象クラスが結合しているクラスの数である. あるクラスが他のクラスのメソッドやインスタンス変数を参照しているとき, 結合しているという.

RFC(クラスの反応;Response For a Class):

計測対象のクラスのメソッドと, それらのメソッドから呼び出されるメソッドの数の和として定義される(すなわち, メッセージに反応して潜在的に実行されるメッセージの数である).

LCOM(メソッドの凝集の欠如;Lack of COhesion in Methods):

計測対象クラス C_i が n 個のメソッド $M_1, ..., M_n$ を持つとする. I_i ($i = 1, ..., n$)を, それぞれメソッド M_i によっ

て用いられるインスタンス変数の集合とする. $P = \{(I_i, I_j) \mid I_i \cap I_j = \phi\}$と定義し, $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \phi\}$と定義する. もし $I_1, ..., I_n$ がすべて ϕ の時は, $P = \phi$ とする. このとき, $LCOM = |P| - |Q|$, ただし, 値が 0 より小さくなるときは 0, と定義する.

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[7].

2.2. 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, Booch 法[3], OMT[12]などの手法が提案されている. 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モデリング/設計記述言語 UML(Unified Modeling Language)が提案された[16]. 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) オブジェクト指向設計仕様作成ツール
(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仕様書作成ツール
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.

3. プロダクト計測ツール

3.1. 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.

3.2. 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 C&K 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. テンプレートを除く C++を対象としている. ツールは C++で実装されている. 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. この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,

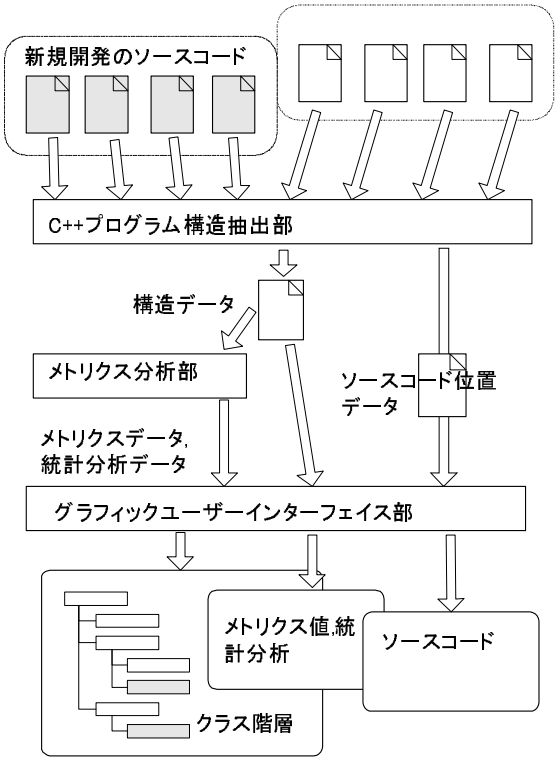

図 1 プロダクト計測ツールのシステム構成
Fig. 1 System structure of product metrics tool

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, C&K 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.

3.3. ツールの機能と特長

ツールは以下の 3 つの情報を GUI により表示する機能を持つ.

(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)C&K 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 C&K 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．

3.4. 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．

(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．

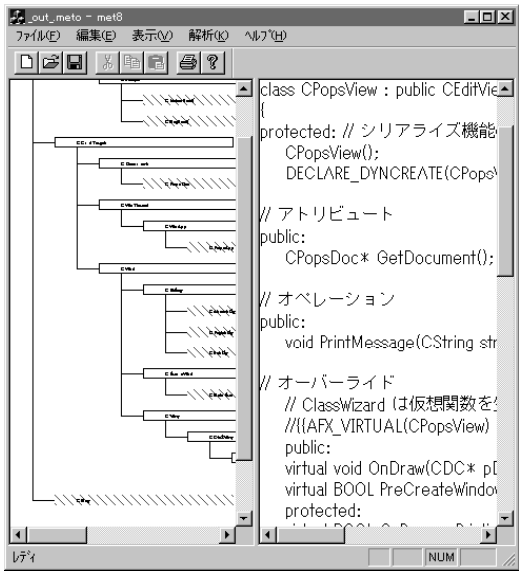

図 4 ソースコードの表示例

Fig. 4 Displaying source code

(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．

(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

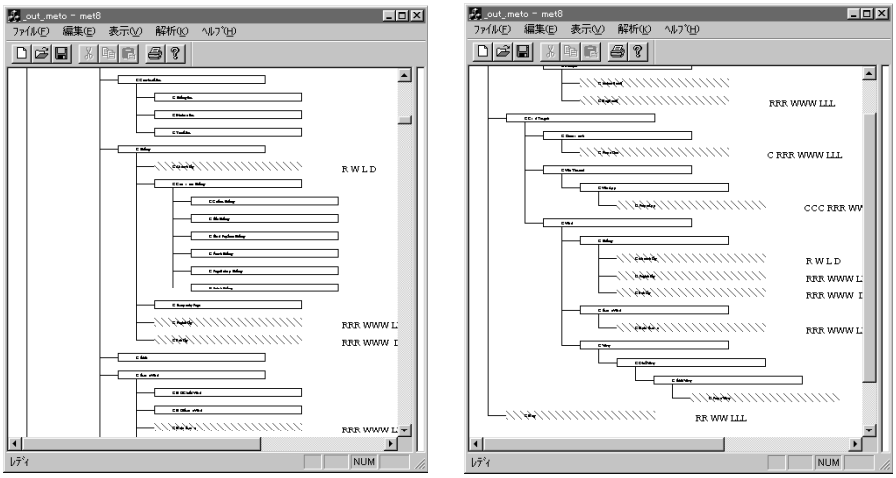

図 2 クラス階層の表示例(a)全体(左), (b)部分(右)

Fig. 2 Sample of class hierachy (a)all (left), (b) partial (right)

れる.

i = min(max(0, (v -

i > 0 の場合は複
にメトリクスの頭文
(図 3 参照).

4. 評価実験

4.1. 概要

本ツールを, 1997
行われた新人研修で

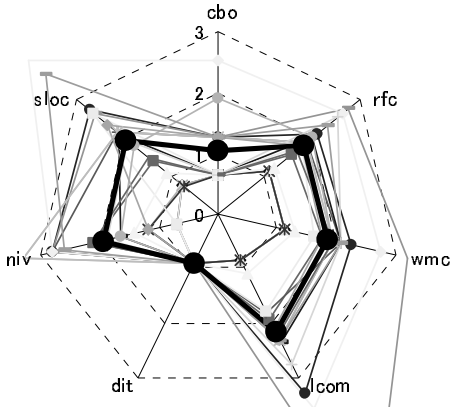

図 5 CDocument 派生クラス(サンプル数 19)
Fig. 5 Classes derived from CDocument (19 samples)

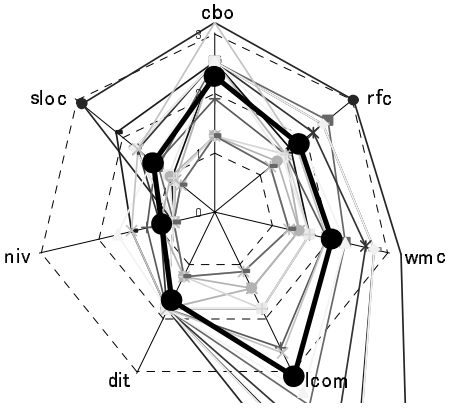

図 6 CView 派生クラス(サンプル数 17)
Fig. 6 Classes derived from CView (17 samples)

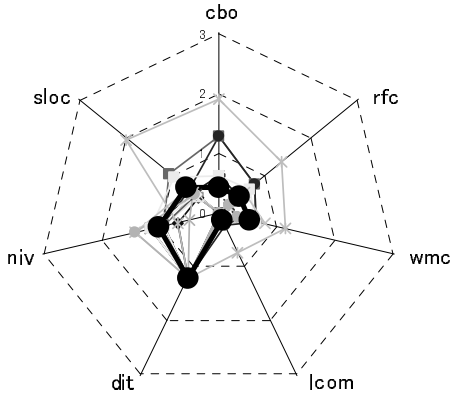

図 7 CDialog 派生クラス(サンプル数 15)
Fig. 7 Classes derived from CDialog (15 samples)

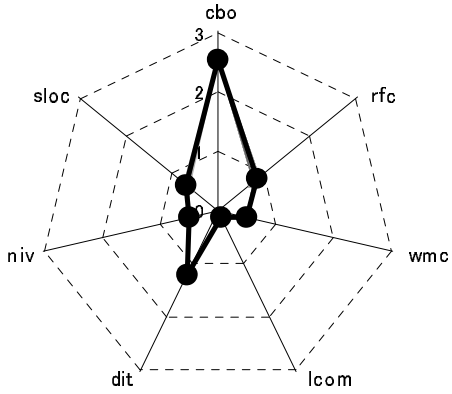

図 8 CWinApp 派生クラス(サンプル数 17)
Fig. 8 Classes derived from CWinApp(17 samples)

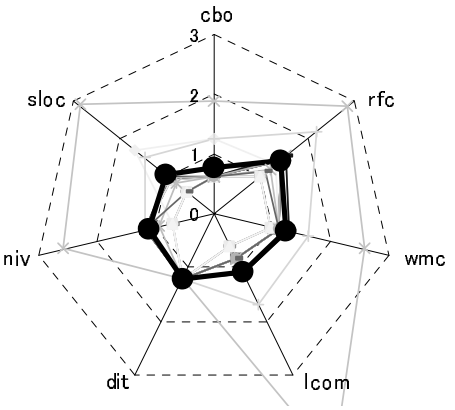

図 9 CFrameWnd 派生クラス(サンプル数 17)
Fig. 9 Classes derived from CFrameWnd (17 samples)

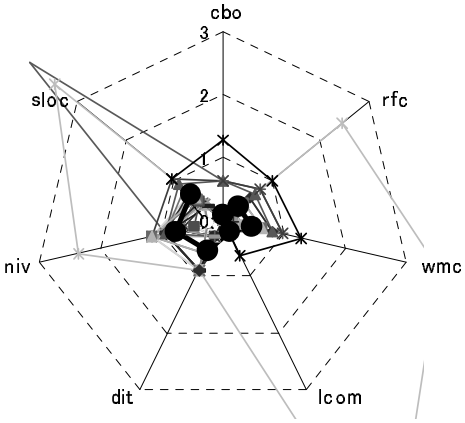

図 10 その他のクラス(サンプル数 39)
Fig. 10 Other classes (39 samples)

4.2. クラスの分類

本実験で作成されたクラスは以下の 6 種類に分類される.

(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(「アプリケーションが前回実行された時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.

4.3. 実験データ

開発された 17 人分のプログラム, 141 のクラス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, その分類に属するクラスすべてのメトリクス値の平均である. CBO, RFC, WMC, LCOM, DIT

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(分)
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
CPassword	1	0	1	0	0	0	0	1	41.4
CConfig	1	0	1	0	0	0	0	0	8.3
CPopServerView	0	0	0	0	0	0	0	0	0
CPopServerApp	0	0	0	0	0	0	0	0	0
CMessageSock	0	0	0	0	0	0	0	1	0.3

i_X はメトリクス X についての異常値, i_{sum} は異常値の合計

Ec はクラスで発見されたフォールトの数, Et はそれらのフォールトを修正するのに要した時間

表 2 異常値とフォールト修正時間の順位相関係数

Table 2 Rank correlation between illegalness and error fixing time

	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 異常値とフォールト個数の順位相関係数

Table 3 Rank correlation between illegalness value and error number

	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

は C&K メトリクス(NOC はすべてのクラスについて 0 であったためグラフには描かれていない), 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. 分析

5.1. メトリクス値とエラーの発生状況

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. 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). 異常値 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.

5.2. 卒業演習で MFC を用いることについての考察

今回の卒業演習において用いられたフレームワークである MFC が, 開発者にどのように理解され, あるいは再利用されたのかを考察する. クラス分類ごとにメトリクス値の分布を調べたことによって, 本卒業演習には以下のような特徴があることがわかった.

(1)フレームワークの設計意図が理解されている

6 つのクラス分類のうち, その他をのぞく 5 分類(CDocument, CView, CDialog, CWinApp, CFrameWnd)は開発ツールがスケルトンコードを生成する. そのため, 開発者が機能を作り込まなければ, メトリクス値はスケルトンコードによる値になる. たとえば, 今回の卒業演習では, 17 人の開発者が CWinApp クラスに全く機能を作り込んでいないため, その分類に属するクラスすべてのメトリクス値が一定となっている(図 8 参照). MFC はドキュメント-ビューアーキテクチャをとっており, ドキュメントクラスにデータの格納と変換操作, ビュークラスにユーザーインターフェイスを作り込むことが前提となっている. グラフによれば, CDocument と CView のメトリクス値の平均値と分散が大きい. これより, 今回の卒業演習では, CDocument と CView 分類にのクラスに多くの機能が作り込まれた。すなわち, 開発者がフレームワークの設計意図を理解し, CDocument と CView に多くの機能を実装していたことがわかる.

(2)派生が活用されていない

作成されたすべてのクラスの NOC が 0 になったということは, 作成されたクラスからさらに他のクラスを派生させた事例がないことを意味する. つまり, 卒業演習で作成されたプログラムは, ライブラリのクラスを再利用していたが, 卒業演習で作成されたプログラムが, さらに再利用されなかったこと(少なくとも継承という形態では)を示している. さらに, 開発ツールが生成するスケルトンのクラスは, 自動的にライブラリのクラスから派生するようになっていたことを考えあわせると, 開発者が意図的に派生を行った兆候は見られない. この原因は, 今回の開発課題で継承を効果的に用いることができるような場面がなかったか, あるいは, 継承という概念の有用性が理解されていなかった, 動機付けが十分ではなかったことが考えられる.

継承はオブジェクト指向開発において, 再利用を行うための重要な機能であることを考えると, 継承させやすい(再利用されやすい)クラスを作ることは教育上重要である. そのため, 継承を使う動機付けが内包されるような演習課題を作り, また, 開発期間を長くして, 再利用を起きやすくするべきである.

(3)ダイアログクラスは曖昧な性格をもつ

MFC を用いて開発されるアプリケーションでは, ユーザーの入力を受け付けたりユーザーに警告を行うために, ダイアログが多用される傾向にある. ダイアログはこのような雑用に用いられる便利なクラスであると同時に, 一通りの機能を備えたウィンドウでもあり, 機能を作り込むことも可能である. そのため, 一般に流通しているソフトウェアのなかには, ダイアログだけでアプリケーションを構築してしまう事例も見受けられる. MFC においては, 通常のウィンドウはドキュメント-ビューアーキテクチャによって設計されるが, ダイアログクラスはこのアーキテクチャーと関わりを持たない.

今回の卒業演習においては, ほとんどのダイアログ

クラスの WMC 値が小さいことから, ダイアログに多くの機能を持たせなかったことがわかる. すなわち, ほとんどのダイアログは単に情報を表示するだけ, あるいは単に入力を受け付けるだけの単純な機能を受け持っていた. しかし, 例外的に多くの機能を作り込まれたダイアログクラスも観察された. MFC を用いた開発の場合, ダイアログクラスにアプリケーションの多くの機能を詰め込むような実装は, フレームワークの設計意図を無視することになる. しいては, わかりにくい設計につながりかねない. 教育現場での対策としては, (1)ダイアログクラスは単にユーザーの入力を受け付けたり, 警告を発するような単純な目的にしか使用しないように指導する, あるいは, (2)そもそもダイアログクラスを実装しないような開発をする, などの対策が必要であると思われる.

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, メトリクスを収集し, ツールの有用性を評価する.

(2)プロダクト評価ツールを拡張する. C&K メトリクス以外のメトリクスや, C++以外のプログラミング言語を扱えるようにする. 上流工程の CASE ツールのデータを扱えるようにする.

(3)メトリクスを改良する. クラス分類をさらにおしすすめて, ツールによる複雑度の判定を, 人間の判定に近くなるようにする.

謝辞

評価実験を協力いただいた, 日本ユニシス株式会社の高橋優亮氏に感謝いたします. メトリクスツールの共同開発者である奈良先端技術大学院大学の高林修司氏に感謝いたします.

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. Basili, 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] IFPUG: *Function Point Counting Practices Manual, Release 4.0*, International Function Point Users Group (1994).

[6] 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).

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

[8] 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).

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

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

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

[12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen: *Object Oriented Modeling and Design*, Prentice Hall (1991).

[13] C. R. Symons: “Function Point Analysis: Difficulties and Improvements”, *IEEE Transaction on Software Engineering*, Vol. 14, No. 1, pp.2-10 (1988).

[14] 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).
[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).
[16] *UML Summary*, ver. 1.1 (1997). (taken from http://www.rational.com/).