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

神谷 年洋・別府 明・楠本 真二・井上 克郎(大阪大学)
毛利 幸雄(日本ユニシス株式会社)

Empirical Evaluation of Complexity Metrics for Object-Oriented Programs

Toshihiro Kamiya, Akira Beppu, Shinji Kusumoto, Katsuro Inoue(Osaka University)
and Yukio Mohri(Nihon Unisys Corporation)

Measuring software product and the development process is essential for improving software productivity and quality. Software metrics are quantitative measures of software products and process. For example, complexity metrics is used to evaluate the difficulty of the maintenance of the program. In order to evaluate the complexity of object-oriented program, several complexity metrics have been proposed. Among them, Chidamber and Kemerer's metrics are the most well-known metrics for object-oriented programs. Also, their effectiveness for estimating the number of faults in a program has been evaluated empirically. However, from the viewpoint of the management, it is more important to estimate the cost to fix the faults in the program rather than the number of ones. This paper empirically evaluates the usefulness of Chidamber's metrics by examining the relationship between the value of the metrics and the cost to fix the faults in the program by using the data collected from actual software development processes. This paper also examines how to apply the metrics to the programs most of which is developed by reusing the classes in the class-library (framework).

キーワード: メトリクス, オブジェクト指向, フレームワーク, 再利用

1 まえがき

近年, ソフトウェアが大規模・複雑化してきたことに伴い, 開発期間の短縮やコストの削減・品質向上の要求が高まっている. そのような要求に応えるためには, ソフトウェアの全ライフサイクル(ソフトウェアの開発・保守)にわたる管理が必要である. “ソフトウェア開発プロセスの品質”という概念は, ソフトウェアを開発するプロセスを改善し, そのプロセスで生産されるソフトウェアの品質を安定させ管理可能にするために生まれたものである[4].

開発プロセスの品質改善のためには, 開発プロセスの各フェーズで開発されるプロダクトの状態を測定し, 分析して, プロセスにフィードバックする必要がある. 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(複雑さ, 信頼性, 効率等)を判別する客観的な数学的尺度である. そのなかでも, ソフトウェアの複雑さメトリクスは, ソフトウェアの品質や保守の容易さを評価／予測するために用いられる. 測定の結果, ソフトウェアが複雑であればあるほど, エラーが含まれている可能性が高く(低品質である), 保守が困難であると評価される.

これまでに提案された代表的なソフトウェア複雑さメトリクスには, Halsted のメトリクス, McCabe のサイクロマチック数などがある[7]. Chidamber と Kemerer は, これらのメトリクスは従来の(非オブジェクト指向の)プログラミング言語で開発されたソフトウェアに対する複雑さメトリクスであり, 最近のオブジェクト指向で開発されたソフトウェアの複雑さを評価するには不十分であると指摘している. そこで, オブジェクト指向で開発されたソフトウェアを対象として, 6 つの複雑さメトリクスを提案している[2]. 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実験的に示されている[1]. しかし, ソフトウェア開発プロセスの管理の観点からは, 単なるエラーの個数の予測よりも, エラーの難易度(例えば, エラー修正に必要な時間)で重み付けたエラー数を予測する方が効果的である[6].

一方, オブジェクト指向のソフトウェア開発では, ソフトウェアを独立性が高く, 組み合わせの容易な部品を利用してソフトウェアを開発することが目的とされている. すでに存在する高品質のソフトウェアを再利用することで品質の向上を実現し, また, 再利用によって開発するソフトウェアの分量を減少させることで開発期間の短縮を目指している. しかし, 文献[1]における Chidamber らのメトリクスの評価では, 積極的な再利用を用いて作成されたソフトウェアに対しては, 有効性の評価が十分に行われていない.

本研究では，Chidamber らのメトリクスを，再利用を積極的に用いて作成されたソフトウェアを対象として適用し，その有効性を評価する．具体的には，教育環境で行なわれたオブジェクト指向ソフトウェア開発プロセスから収集したデータを用いて，メトリクスの値とエラーの難易度で重み付けをしたエラー数との関係を評価する．また，フレームワークを用いた再利用度の高いソフトウェアに対する Chidamber らのメトリクスの適用方法についても検討する．

以降，2.では準備としてオブジェクト指向開発の特徴と Chidamber らのメトリクスを紹介する．3.では研究の目的を明らかにし，実験の概要を説明する．4.では実験データに対する分析を行う．最後に 5.では，まとめと今後の課題を示す．

2 オブジェクト指向ソフトウェア開発

2.1 オブジェクト指向開発の特徴　従来の開発では，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”を使用してきた．ライブラリを用いた開発では，プログラム全体の処理の流れなどの主要な部分は開発者が開発し，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．

一方，オブジェクト指向開発における再利用では，“フレームワーク”と呼ばれる特殊なライブラリが用いられる．フレームワークとは，開発対象となるドメイン固有のソフトウェアアーキテクチャを抽出し，その特徴を構造に反映したクラスライブラリであり，一定の設計思想によって組み立てられたクラスの結合体である[3]．フレームワークを用いた開発では，プログラム全体の処理の流れなどの主要部分をフレームワークから取り出し，新たに必要な部分を開発者が開発して，それを組み合わせることになる．

GUI ドメインは，フレームワークの実用化がもっとも進んでいるドメインであり，MFC(Microsoft Foundation Class)等の商品化されたフレームワークが存在する[3]．3.で述べる評価実験でも MFC が用いられる．

2.2 オブジェクト指向複雑さメトリクス　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さメトリクスであり，クラスの定義からその複雑さを測定する．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さであると仮定できるときは，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，評価対象のクラスの，木の中での深さを示す．
- 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．

Basili らの実験によると，これらのメトリクスが，オブジェクト指向開発プログラムに対して，従来のメトリクスよりも正確にエラー数を予測することが確認されている[1]．しかし，エラーの修正に要する時間は，エラーの種類によって異なるため，開発管理上，エラーの個数だけを予測することは不十分である．複雑さの評価においては単なるエラー数を用いるのではなく，エラーの難易度で重み付けを行うほうが適当である．

3 評価実験

3.1 実験目的　本研究の目的は，フレームワークを用いて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，Chidamber らのメトリクスの有効性の評価を，エラーの難易度で重み付けしたエラー数との関係を用いて行うことである．具体的には，先ず，オブジェクト指向プログラムに特有なエラーを整理・分類し，エラーの修正時間で重み付けをしたエラー数を求める．次に，Chidambar らのメトリクスの値と，エラーの修正時間で重み付けしたエラー数との関係を調べる．最後に，フレームワークから多くのクラスを再利用して開発されたプログラムに対して Chidamber らのメトリクス適用する方法を述べ，その結果を考察する．

3.2 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，オブジェクト指向開発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，開発プロセスは，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から構成される．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1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1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.

3.3 データの収集方法 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)上で作業を行う. 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1時間おき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. これらのプログラムソースファイルに基づいてメトリクスの値を算出する.

一方, 開発プロセスやエラーデータを追跡するために, 種々の報告書を用意した. 報告書は作業分担表, レビュー報告書, 単体および結合テスト報告書, エラー特定報告書, エラー修正報告書, である. 更に, 開発プロセスのコーディング, レビュー, 単体テスト, 結合テストの各フェーズの終了時点のプログラムソースファイルを, 被験者がプログラムリストとして提出した.

3.4 収集データ(1) エラーデータ 本研究では, コードレビュー以降に発見されたエラーを対象に分析を行う. エラーの修正に要した時間を測定するため, 報告書には時刻を記入する欄が設けられていた. また, エラーを分類し記入するための欄も設けてあり, これを基にしてエラーを次の9種類に分類した.

(E1)クラス階層:クラス階層の設計の誤り
(E2)メソッド不足:機能の不足
(E3)インターフェイス:入出力データの条件(範囲)やフォーマットの誤り, 引数や返値の宣言誤り
(E4)メンバ:インスタンス変数(オブジェクトの属性)や局所変数の誤りや不足
(E5)制御:ループの条件判定などの誤り
(E6)場合分け:if文やswitch文の誤り
(E7)初期化:初期化の誤り
(E8)誤解:言語, 処理系, ライブラリに対する誤解
(E9)その他

エラー分類と各エラーの修正に要した時間の関係を, 表1に示す.

3.5 収集データ(2) メトリクスの値 一方, プログラムの複雑度メトリクスの値は, 上記のエラーの定義と矛盾しないように(すなわち, 複雑度メトリクスの値とそのプログラムに含まれている重み付きエラー数との関係を調べるために), コードレビュー直前のプログラムソースファイルから算出した(文献[2]に従い, フレームワークから再利用された部分と, 開発者が新たに開発した部分の両方を対象としている).

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.

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.

表2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Chidamberらの6種の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いて, 値が0であったため省略している.

4 分析

4.1 エラー分析 表1より, エラー分類ごとに発生頻度や修正に要した時間に大きな差が見られる.

例えば, (E1)クラス階層のエラーと(E5)制御のエラーは, 他のエラーと比較して修正に要した時間が長いことが観察される(それぞれ260分と241分である). クラス階層のエラーの修正に時間を要したのは, 複数のクラスの定義を修正しなければならないためである. また, 制御のエラーの個々の事例を調べた結果, そのすべてがWindowsのイベント処理の部分で発生していた. つまり, 制御のエラーの修正に時間がかかったのは, 被験者の大部分がMFCに不慣れであったこと, および, Windowsのイベント(クリックやメニューの選択など)が, 複雑な方法でクラスのメソッドとマッピングされていることが理由であると考えられる.

一方, (E3)インターフェイスのエラーは, 個数は多い(22個)が, その修正に要した時間は短いことが確認される.

以上のように, エラーの種類によって修正に要する時間が異なるという結果は, 単なるエラーの数ではなく, エラーの修正に要する時間, すなわちエラーの難易度をプログラムの品質の評価に用いるのが望ましいことを表している.

4.2 メトリクスと重み付けしたエラー数 各被験者が作り込

んだエラー数と修正時間で重み付けをしたエラー数(表 2)より，メトリクスの値とエラーの数，エラーの修正に要した時間との相関を求めた(表 3)（これらの間に相関があることは有意水準 1%で検定されている[5]). これより，エラー数よりも，エラーの難易度(エラーの修正に要した時間)の方が，メトリクスの値と相関が高いことがわかる. とくに，DIT が 0.77 と高い相関をもっている. 一方，RFC は，これらの中では 0.54 と低い値となっている.

以上の結果より，Chidamber らのメトリクスは，エラーの難易度を考慮したエラー数の予測にも有効であることが確認された.

4.3 再利用の影響　図 1 に示すように，今回の開発では，フレームワークを再利用した部分が，新規開発された部分に比較して大規模である. 一般に再利用されたプログラムは新規開発部分よりも品質が高く，それがメトリクスに影響を及ぼすことが観察されている[1]. そこで，再利用したクラスの影響を受けるメトリクス CBO，RFC に関して，再利用を考慮して適用することを考える.

CBO は，2.2 で述べたように，評価対象のクラスが“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属性を参照することにより結合する場合，そのクラスを再利用するためには，クラスの定義を知っていなければならない. 一方，RFC は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. メソッドを介してのみ他のクラスを参照する場合，そのクラスのを再利用するのに，そのクラスの内部の定義を知る必要はない.

そこで，フレームワークから再利用した部分と新規開発部分を区別して，これらのメトリクスの値を求めてみる(表 4). CBo と RFCo を次のように定義する:

- CBOo:評価対象のクラスが結合しているクラスの中で，開発者が開発したクラスに対する結合の数.
- RFCo: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中で，開発者が開発したクラスのメソッドであるものの集合の和集合の要素数.

表 4 より，CBO と CBOo，RFC と RFCo の値にはかなりの差が見られる. CBOo と RFCo とエラーの修正に要した時間との相関を表 5 に示す.

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. すなわち，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さに寄与していることになる. 今回の実験では，相手のクラスの属性を直接参照するような“結合”が多く見られた. このような結合を行う場合には，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フレームワークのクラスであっても，開発者が新規に開発したクラスと同様に，複雑さを増大させると考えられる.

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. すなわち，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さに寄与していない. フレームワークのメソッドの定義は，開発期間を通じて変更されないため，フレームワークのクラスをメソッドのみを参照して再利用する場合，定義を知ることなく使用できる. 一方，開発者が新規に作成するメソッドの定義は開発中に何度も変更されるため，その定義を知っていなければならない. これが，開発者が作成するメソッドを参照することがより複雑さを増大させる理由であると考えられる.

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.

5 まとめ

本研究では，Chidamber らのメトリクスがエラー修正時間と相関があることを実験的に示した. また，フレームワークを用いた開発において，クラスの再利用が Chidamber らのメトリクスの CBO と RFC に影響を与えることを明らかし，その適用方法について検討した.

今後の課題としては，より大規模な開発を対象としてメトリクスの評価を行うことや，クラスライブラリの再利用の度合いを考慮に入れた複雑度メトリクスの開発を検討している.

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. 20, No. 22, 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] 本位田信一・青山幹雄・深澤良彰・中谷多賀子:”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-”, 情報処理学会論文誌, 31, 12, pp.1812-1821(1991)
[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)

神谷 年洋　(非会員) 平 8 年 3 月大阪大学基礎工学部中退. 同年, 同大学院基礎工学研究科情報数理系専攻博士前期課程入学.. 現在に至る. ソフトウェア開発プロセスの管理とオブジェクト指向開発に関する研究に従事.

別府 明　(非会員) 平 9 年大阪大学基礎工学部情報工学科卒業. 同年, 同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入学, 現在に至る. ソフトウェアメトリクスに関する研究に従事.

楠本 真二　(非会員)昭 63 大阪大学基礎工学部情報工学科卒業. 平 3 同大大学院博士課程中退. 同年同大基礎工学部情報工学科助手. 平 8 同大講師, 現在に至る. 工学博士. ソフトウェアの生産性や品質の定量的評価, プロジェクト管理に関する研究に従事. 平 5 年電子情報通信学会論文賞受賞. 電子情報通信学会, 情報処理学会, IEEE 各会員.

井上 克郎　(非会員)昭 54 大阪大学基礎工学部情報工学科卒業. 昭 59 同大学大学院博士課程修了. 同年同大基礎工学部情報工学科助手. 同年 8 月ハワイ大学マノア校芸術・科学学部助教授. 平元年大阪大学基礎工学部講師. 平 3 年 11 月同大助教授. 平 7 年 12 月同大教授, 現在に至る. 工学博士. ソフトウェア工学の研究に従事. ACM, IEEE, 電子情報通信学会, 情報処理学会各会員.

毛利 幸雄　(非会員)昭 49 年青山学院大学理工学部卒業. 同年日本ユニシス株式会社入社. 現在, 総合教育部所属. ソフトウェア教育, オブジェクト指向開発に関する研究に従事.

表 1 エラー分類

Table 1. Error classification

分類	Ec	Et	AVEt
E1	2	520	260
E2	8	450	56
E3	22	2000	91
E4	13	1355	104
E5	3	722	241
E6	3	36	12
E7	6	396	66
E8	5	470	94
E9	2	55	28
合計	64	6004	94

Ec: エラー数

Et: エラーの修正に要した時間(分)

AVEt; エラー1 個当たりの修正に要した時間(分)

表 2 Chidamber らのメトリクスの値との修正に要した時間

Table 2. Chidamber's metrics, numbers of errors, and error fixing efforts

被験者	WMC	CBO	RFC	LCOM	DIT	Ec	Et
t1	33	38	75	73	17	7	1124
t2	19	16	38	47	10	2	50
t3	22	21	58	46	14	5	315
t4	7	8	14	16	6	0	0
t5	19	17	41	47	10	2	390
t6	8	8	13	14	6	2	114
t7	19	17	40	47	10	3	21
t8	20	18	32	49	14	7	891
t9	8	9	16	13	6	0	0
t10	25	24	58	62	16	5	530
t11	21	18	52	52	12	8	576
t12	24	20	50	59	16	8	1005
t13	8	9	16	13	6	1	60
t14	38	37	90	88	20	4	850
t15	22	20	55	55	12	3	154
t16	26	23	67	57	16	1	94
t17	11	10	24	11	6	1	90
t18	17	13	24	47	10	3	75
t19	8	9	15	13	6	1	25

表 3 Chidamber らのメトリクスについての相関係数

Table 3. Correlation coefficient between Chidamber's metrics and number of errors/error fixing efforts

	WMC	CBO	RFC	LCOM	DIT
Ec	0.62	0.58	0.54	0.65	0.68
Et	0.72	0.74	0.63	0.70	0.77

表 4 CBOo と RFCo

Table 4. CBOo and RFCo

被験者	CBOo	RFCo	Et
t1	8	31	1124
t2	3	17	50
t3	3	19	315
t4	0	9	0
t5	3	17	390
t6	0	9	114
t7	2	17	21
t8	0	24	891
t9	0	8	0
t10	2	24	530
t11	1	19	576
t12	1	22	1005
t13	0	8	60
t14	3	35	850
t15	3	20	154
t16	3	24	94
t17	0	10	90
t18	0	16	75
t19	0	5	25

表 5 CBOo と RFCo に関する相関係数

Table 5. Correlation coefficient between CBOo/RFCo and errors/error fixing efforts

	CBOo	RFCo
Et	0.47	0.77