The University of Osaka
Institutional Knowledge Archive

Title	複雑度メトリクスを用いたエラー予測の一手法 : アプリケーションフレームワークを用いた開発への適用
Author(s)	神谷, 年洋; 楠本, 真二; 井上, 克郎 他
Citation	情報処理学会論文誌. 2001, 42(6), p. 1601-1609
Version Type	VoR
URL	https://hdl.handle.net/11094/50161
rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

複雑度メトリクスを用いたエラー予測の一手法 ——アプリケーションフレームワークを用いた開発への適用

神谷年洋[†] 楠本真二[†]
井上克郎[†] 毛利幸雄[††]

本論文では，アプリケーションフレームワークに基づくオブジェクト指向プログラムに含まれるクラスを対象として，複雑度メトリクスによるエラーの予測精度を向上させることを目的とした，クラスを分類する方法の提案および実験的評価を行う．実験では，Chidamber らのメトリクスを含むいくつかの複雑度メトリクスを用いて，アプリケーションフレームワーク MFC（Microsoft Foundation Class）を用いて開発された C++プログラム中のクラスのエラー修正時間の予測を行い，本手法により予測精度が改善することが示された．

An Error Estimation Method for C++ Program with Object-oriented Applicatoin Framework

Toshihiro Kamiya,[†] Shinji Kusumoto,[†] Katsuro Inoue[†] and Yukio Mohri[††]

For improvement precision of error estimation by complexity metrics, this paper propses a method to classify classes in object-oriented programs developed with an application framework and evaluates the method emprically. In the experiment, the method improves precision for Chidamber and Kemerer's metrics to estimate time to fix errors of each class in a programs develped with a application framework MFC (Microsoft Foundation Class).

1. まえがき

近年，ソフトウェアの応用分野の拡大とともに，ソフトウェアが大規模・複雑化してきている．それにともない，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に基づいた分析・設計法，プログラミング言語などが数多く提案され，実際の開発現場でも使われるようになってきている[1)]．

また，オブジェクト指向開発の普及にともない，設計やドメインに関する知識が蓄積されてきたことにより，ドメインを限定することで大規模な再利用を可能とするフレームワークやコンポーネントといったライブラリが登場してきた．また，デザインパターンやビジネスオブジェクトといった分析・設計を補助するための手法も提案されてきている．これらのアプローチは，ソフトウェアの開発時に生産性を向上させるためのアプローチである．

一方，ソフトウェアメトリクスを用いた生産性や品質向上のアプローチも広く受け入れられている．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ソフトウェアの状態を評価することで，問題の含まれる部分に対する変更を行ったり，その部分に対するレビュー・テスト工数の割当を増やしたりするという対処が施される．メトリクスを用いたアプローチを実行する際には，メトリクス値を計測するためのツール

† 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
Nihon Unisys, Ltd.

と計測結果を用いたエラー☆予測の手法の確立が必要である．

複雑度メトリクスを用いて，クラスにエラーが含まれているかどうか，クラスに含まれているエラー数が何個か，クラスに含まれるエラーの修正工数，を予測する手法がある．その方法は一般に，(1) メトリクスおよびエラーに関するデータを収集し，(2) 予測式を作成し，(3) 予測式を適用してエラーを予測する，の 3 つの段階からなる．クラスの種類によって予測式を別々に立てるほうがよいという指摘[2] はあるが，クラスの種類分けを自動的に行う一般性のある方法は難しく，実際にそのような方法を用いてエラー予測精度が向上したという報告はない．

そこで，本論文では，アプリケーションフレームワークを用いた開発に限定することで，1 つの分類方法を示し，上述の方法で予測する実験を行って，有効性を評価した．実験の結果，CK メトリクスを用いてエラーが含まれるクラスを予測する精度が，本手法によって改善されたことが確認できた．

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[9]．

たとえば，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[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] などがある．

オブジェクト指向ソフトウェアのクラス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（CK メトリクス）を提案している[5]．CK メトリクスは，オブジェクト指向ソフトウェア向けのメトリクスとして有名であり，複数の研究者による複数の実験に用いられていて[4),14]，オブジェクト指向ソフトウェアに対しては従来のコードメトリクスより有効であるとの報告がなされている[2]．

以下に CK メトリクスの定義を示す．これらの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している．

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_i が，メソッド $M_1, ..., M_n$ を持つとする．これらのメソッドの複雑さをそれぞれ $c_1, ..., c_n$ 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 5) 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ている．本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．$P = \{(I_i, I_j) | I_i \cap I_j = \emptyset\}$ と定義し，$Q = \{(I_i, I_j) | I_i \cap I_j \neq \emptyset\}$ と定義する．もし $I_1, ..., I_n$ がすべて $\emptyset$ のときは，$P = \emptyset$ とする．このとき，$LCOM = |P| - |Q|$，ただし，$LCOM < 0$ のときは，$LCOM = 0$ 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

☆ IEEE 標準では，ソフトウェアの開発作業において人間がおかす誤りをエラー，エラーがソフトウェア中に具体化したものをフォールトと定義している．本論文中ではエラーとフォールトを同じ意味に用いる．

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，クラスごと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[5]．Basili らは，CK メトリクスを用いて効果的にエラー発生が予測できることを示した．CK メトリクスが複雑であると判定したクラスは，そうではないクラスよりもエラー発生確率が高いことを，実験データによって示した[2]．

我々も，アプリケーション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，CBO と RFC を修正したメトリクスを定義した[8]．それらを次に示す．

CBON，CBOR：CBOR と CBON は新規開発クラスに対してのみ定義される．CBOR は計測対象クラスが結合している再利用された（すなわち，ライブラリに含まれる）クラスの数である．CBON は計測対象クラスが結合している新規開発クラスの数である．定義より，任意の新規開発クラス c に対して，$CBON(c)+CBOR(c) = CBO(c)$ となる．

RFCN，RFCR：RFCN と RFCR は新規開発クラスに対してのみ定義される．RFCN は計測対象のクラスのメソッド数と，それらのメソッドから呼び出されるメソッドのうち新規開発クラスに属するものの数の和である．RFCR は計測対象のクラスのメソッドから呼び出されるメソッドのうち再利用クラスに属するものの数の和である．定義より，任意の新規開発クラス c に対して，$RFCN(c) + RFCR(c) = RFC(c)$ となる．

2.2 エラー発生予測手法

一般に，複雑度メトリクスを用いてクラスのエラー発生を予測する手順は一般に次のようになる．

(1) 基準となるデータの収集：CK メトリクスを計測するためのソースコードあるいはクラスの設計書，発見されたエラーとエラーごとに修正されたクラスの記録を収集する．

(2) 予測式の作成：メトリクスの計測値から統計分析によって，クラスにエラーが含まれるかどうか，あるいは，含まれる数，エラー修正労力を予測する式を作る．たとえば，Basili らの実験では，クラスのメトリクス値を入力とし，エラーの有無（2 値）を出力としていたため，多変量ロジスティック回帰分析を用いている．

(3) エラー発生の予測：計測対象クラスから得られたメトリクス値をエラーの予測式に適用し，エラーの有無や数，修正労力を予測する．

クラスの種類（たとえば，ユーザインタフェー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エラー予測の有効性に大きな違いがあることが指摘されている[2]．したがって，クラスの種類ごとに異なる予測式を用いることで予測精度が向上することが期待されるが，任意の開発においてクラスの種類分けをソースコードから自動的に行うことは困難である．

3. クラス分類を用いた予測手法

アプリケーションフレームワークを用いた開発においては，フレームワークのクラス階層は（厳密ではないにしても）クラスの種類を反映すると考えられる．また，フレームワークに含まれるクラスから導出によって新しいクラスを作成することが多く行われる．そのような開発において，クラスの種類の代わりに，そのクラスが導出されたフレームワークの親（または先祖）クラスを用いる．クラス分類によって予測式のパラメータ（係数）を変更することにより，エラー予測精度の向上が期待できる．クラス分類はフレームワークのクラス階層に依存するため，フレームワークごとに係数を用意する必要がある．クラス分類と CK の複雑度メトリクスを用いてクラスのエラー発生を予測する手順は次のようになる．

(1) 分類クラスの選出：フレームワークのクラスから，分類に適したクラス（以下「分類クラス」）を選出する．フレームワークのアーキテクチャを考慮して，代表的なクラスを選ぶ必要がある．

(2) 基準となるデータの収集：CK メトリクスを計測するためのソースコードあるいはクラスの設計書，発見されたエラーとエラーごとに修正されたクラスの記録を収集する．

(3) 予測式の作成：クラスの親（または先祖）クラスがどの分類クラスであるかによって，収集データのクラスを分類する．それらの分類ごとに，メトリクスの計測値から統計分析によってエラーの有無や数，修正労力を予測する式を作る．

(4) エラー発生の予測：クラスに対して，分類に従って予測式を適用し，エラーの有無や数，修正労力を予測する．

一般に，フレームワークのクラス間にも継承関係があるため，分類クラス間にも継承関係が生じる．子クラスは親クラスを特化したクラスであると考えられるため，分類としては子（あるいは子孫）側のクラスを用いるべきである．**図 1** において，分類クラス B は A の子孫であるため，新規開発クラス q と u の分類

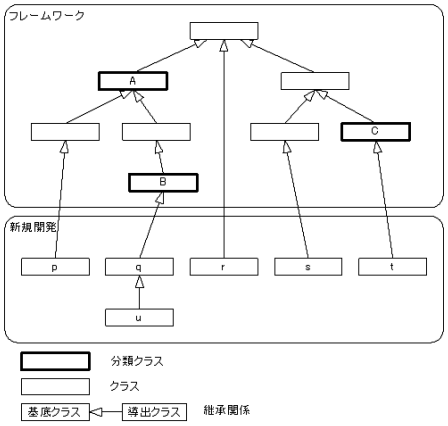

図 1 分類クラスの例
Fig. 1 An example of classification.

表 1 分類クラス例における分類
Table 1 Classification in the example.

Class	Classification
p	A
q	B
r	その他
s	その他
t	C
u	B

は B となる（**表 1**）.

また，C++などの多重継承を許すプログラミング言語を使用した場合には，分類クラス間で継承関係がなくても，あるクラスの分類クラスが複数になる可能性がある．もしこのような状況が発生すると想定される場合には，(1) あらかじめ分類クラス間の優先順位を指定する，あるいは (2) 多重継承しているクラスは別の分類とする，などの対策が必要である．

4. 評価実験

4.1 概　　要

提案した手法を，ある企業で行われた新人研修におけるプログラム開発プロジェクトで得られたデータに適用した．このプロジェクトでは，電子メールの配送システムを開発する．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．プロジェクト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．最終的にインストラクタによる受け入れテストが実施される．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 を用いる．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．

表 2 分類 CDialog のメトリクスおよびエラー
Table 2 Metrics and errors for CDialog.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
1	0/0	2/2	2	1	4	0	3	44	0	0
2	2/0	7/4	4	0	4	0	4	111	0	0
3	0/0	2/2	2	1	4	0	4	47	0	0
4	1/0	4/3	3	3	4	0	2	56	0	0
5	0/0	2/2	2	1	4	0	5	50	0	0
6	2/0	7/4	4	6	4	0	4	66	0	0
7	0/0	2/2	2	1	4	0	4	49	0	0
8	1/0	4/3	3	3	4	0	5	56	0	0
9	0/0	2/2	2	1	4	0	5	51	0	0
10	0/0	2/2	2	1	4	0	5	51	0	0
11	1/0	6/4	4	4	4	0	5	101	0	0
12	0/0	4/4	4	2	4	0	4	60	0	0
13	0/0	5/5	5	0	4	0	6	68	0	0
14	3/0	12/7	7	19	4	0	3	204	2	82.6
15	0/0	2/2	2	1	4	0	6	53	0	0

CLS#はクラスの通し番号．CBO/R の欄には CBO と CBOR の値が示されている．CBON の値は $CBON = CBO - CBOR$ によって求める．RFC，RFCR，RFCN も同様．EC はクラスに発見されたエラーの数，ET はそれらの修正に要した時間．

表 3 分類 CDocument のメトリクスおよびエラー
Table 3 Metrics and errors for CDocument.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
16	1/0	10/7	7	21	3	0	3	78	0	0
17	1/0	14/11	11	49	3	0	9	144	0	0
18	1/0	10/7	7	21	3	0	3	78	0	0
19	1/0	10/7	7	21	3	0	5	79	0	0
20	1/0	10/7	7	21	3	0	5	77	0	0
21	2/1	19/14	14	83	3	0	12	284	1	9.27
22	2/0	17/13	13	46	3	0	12	223	0	0
23	1/0	18/13	13	60	3	0	7	217	1	0.27
24	2/1	25/20	20	148	3	0	12	272	6	168
25	4/1	24/17	17	90	3	0	12	420	2	255
26	1/0	16/10	10	45	3	0	3	276	1	0.1
27	1/0	11/8	8	28	3	0	6	113	0	0
28	2/1	17/12	12	58	3	0	14	224	3	135
29	2/1	17/12	12	58	3	0	14	224	1	7.72
30	3/1	18/12	12	58	3	0	7	186	2	38.6
31	1/0	23/13	13	70	3	0	8	201	1	20.4
32	1/0	15/12	12	56	3	0	9	155	5	68.6
33	2/0	21/13	13	58	3	0	11	381	1	0.1
34	2/1	16/11	11	53	3	0	5	244	0	0

4.2 クラス分類

フレームワーク MFC のアーキテクチャを考慮し，分類クラスとして以下の 5 つを選定した．

(1) CDocument：派生クラスに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：派生クラスにはユーザに対してデータを表示する部分が主に記述される．
(3) CDialog：派生クラスには，ユーザからデータを受け取る部分と，ユーザ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：派生クラスには，アプリケーションの設定に関する処理（「アプリケーションが前回実行されたとき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：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

表 4 分類 CView のメトリクスおよびエラー
Table 4 Metrics and errors for CView.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
35	4/1	15/10	10	45	6	0	3	94	0	0
36	4/1	22/14	14	91	6	0	3	127	0	0
37	4/1	14/9	9	36	6	0	3	89	0	0
38	4/1	14/9	9	36	6	0	3	88	0	0
39	4/1	19/16	16	114	6	0	4	173	0	0
40	5/1	27/20	20	190	6	0	3	300	7	43.3
41	3/1	14/12	12	64	6	0	5	133	0	0
42	4/1	21/17	17	134	6	0	6	220	0	0
43	4/1	21/17	17	136	6	0	3	179	0	0
44	4/1	21/17	17	136	6	0	3	179	0	0
45	4/1	13/10	10	45	6	0	3	105	0	0
46	4/1	17/14	14	85	6	0	7	140	1	86.1
47	5/1	16/12	12	66	6	0	6	173	1	0.43
48	2/1	11/8	8	28	4	0	3	76	0	0
49	2/1	12/9	9	36	4	0	4	98	7	40.6
50	2/1	14/10	10	45	4	0	3	93	0	0
51	2/1	11/8	8	28	4	0	3	78	0	0

表 5 分類 CWinApp のメトリクスおよびエラー
Table 5 Metrics and errors for CWinApp.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
52	4/3	7/3	3	3	4	0	2	68	0	0
53	4/3	7/3	3	3	4	0	2	68	0	0
54	4/3	8/3	3	3	4	0	2	74	0	0
55	4/3	7/3	3	3	4	0	2	66	0	0
56	4/3	8/3	3	3	4	0	2	74	0	0
57	4/3	8/3	3	3	4	0	2	74	0	0
58	4/3	8/3	3	3	4	0	2	74	0	0
59	4/3	8/3	3	3	4	0	2	72	0	0
60	4/3	8/3	3	3	4	0	2	74	0	0
61	4/3	7/3	3	3	4	0	2	68	0	0
62	4/3	8/3	3	3	4	0	2	74	0	0
63	4/3	8/3	3	3	4	0	2	74	0	0
64	4/3	8/3	3	3	4	0	2	78	0	0
65	4/3	8/3	3	3	4	0	2	72	0	0
66	4/3	8/3	3	3	4	0	2	74	0	0
67	4/3	8/3	3	3	4	0	2	74	0	0
68	4/3	7/3	3	3	4	0	2	68	0	0

表 6 分類 CFrameWnd のメトリクスおよびエラー
Table 6 Metrics and errors for CFrameWnd.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
69	1/0	13/7	7	21	4	0	5	99	0	0
70	1/0	13/7	7	21	4	0	5	97	0	0
71	1/0	13/7	7	21	4	0	5	97	0	0
72	1/0	14/8	8	28	4	0	5	101	0	0
73	1/0	13/7	7	21	4	0	5	105	0	0
74	1/0	9/6	6	15	4	0	3	60	0	0
75	1/0	13/7	7	21	4	0	5	99	0	0
76	1/0	14/8	8	28	4	0	5	102	0	0
77	1/0	9/6	6	15	4	0	3	60	0	0
78	2/0	12/7	7	21	4	0	3	178	3	600
79	1/0	9/6	6	15	4	0	3	63	0	0
80	1/0	9/6	6	15	4	0	3	63	0	0
81	3/0	26/16	16	116	4	0	11	302	17	58.3
82	1/0	11/7	7	21	4	0	4	84	0	0
83	1/0	13/7	7	21	4	0	5	100	0	0
84	2/0	20/10	10	43	4	0	5	156	1	0.23
85	1/0	10/7	7	21	4	0	5	63	0	0

CFrameWnd 派生クラスに記述される（ユーザインタフェースが複雑になると，複数のビューを切り替える方法はよく用いられる）．

これらのいずれからも派生しないクラスは，(6) その他に分類される．

今回の実験では，選出された分類クラス間に継承関係は存在せず，複数の分類クラスを継承するクラスが定義されることもなかった．

表 7 分類 CSocket のメトリクスおよびエラー
Table 7 Metrics and errors for CSocket.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
86	0/0	2/2	2	1	3	0	1	51	0	0
87	0/0	1/1	1	0	3	0	1	41	0	0
88	0/0	1/1	1	0	3	0	1	40	0	0
89	0/0	2/2	2	0	3	0	1	46	1	0.3
90	0/0	22/22	22	157	3	0	10	361	1	1.25
91	0/0	2/2	2	0	3	0	5	63	1	0.68
92	0/0	3/3	3	1	3	0	5	71	0	0
93	0/0	0/0	0	0	3	0	0	31	0	0
94	0/0	2/2	2	1	3	0	4	43	0	0
95	0/0	1/1	1	0	3	0	4	42	0	0
96	0/0	1/1	1	0	3	0	4	43	0	0
97	0/0	1/1	1	0	3	0	1	41	0	0
98	0/0	1/1	1	0	3	0	1	39	0	0
99	0/0	1/1	1	0	3	0	4	45	0	0
100	0/0	1/1	1	0	3	0	4	45	0	0
101	0/0	1/1	1	0	3	0	4	47	0	0
102	0/0	5/5	5	4	3	0	4	82	0	0
103	0/0	0/0	0	0	3	0	4	35	0	0
104	0/0	5/5	5	4	3	0	4	73	0	0

表 8 分類その他のメトリクスおよびエラー
Table 8 Metrics and errors for Others.

CLS#	CBO/R	RFC/R	WMC	LCOM	DIT	NOC	NIV	SLOC	EC	ET
105	0/0	3/3	3	3	1	0	3	57	0	0
106	0/0	3/3	3	3	1	0	2	59	0	0
107	0/0	3/3	3	3	1	0	2	59	0	0
108	0/0	3/3	3	3	1	0	5	61	0	0
109	0/0	3/3	3	3	1	0	3	39	0	0
110	0/0	3/3	3	3	1	0	3	57	0	0
111	1/0	7/6	6	0	1	0	4	416	3	0.62
112	0/0	3/3	3	3	1	0	3	57	0	0
113	0/0	3/3	3	3	1	0	3	57	0	0
114	0/0	3/3	3	3	0	0	2	50	1	8.3
115	1/0	6/5	5	0	0	0	4	94	8	42.5
116	1/0	7/6	6	0	0	0	5	109	0	0
117	2/0	9/8	8	16	0	0	1	107	1	77.9
118	0/0	3/3	3	3	0	0	1	63	1	41.4
119	0/0	0/0	0	0	0	0	5	7	0	0
120	0/0	0/0	0	0	0	0	5	7	0	0
121	0/0	0/0	0	0	0	0	3	5	0	0
122	0/0	0/0	0	0	0	0	3	5	0	0
123	0/0	4/4	4	6	2	0	5	53	0	0
124	0/0	4/4	4	6	2	0	5	57	0	0

4.3 実験データ

最終的には，17 人分のデータが利用できた．開発された 17 人分のプログラム，124 のクラスから抽出したメトリクス値，およびエラー個数，修正時間を**表 2**〜**8** に示す．メトリクス値のレーダチャートを**図 2**〜**7** に示す．各グラフの，細い線で描かれた 1 つの多角形が，1 つのクラスについてのメトリクス値を表す．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 は CK 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．分類の平均値（太い線）のメトリクス値の傾向は大きく異なっている．また，

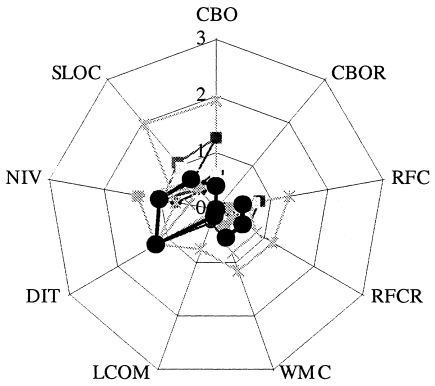

図 2 分類 CDialog のメトリクス値
Fig. 2 Metric values for CDialog classes.

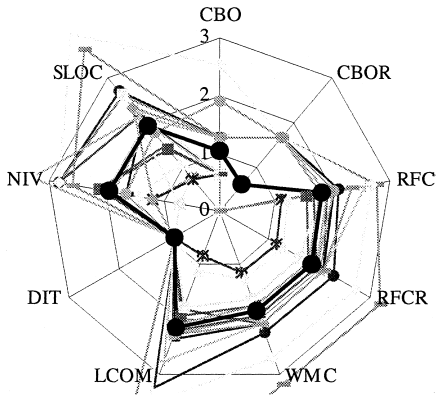

図 3 分類 CDocument のメトリクス値
Fig. 3 Metric values for CDocument classes.

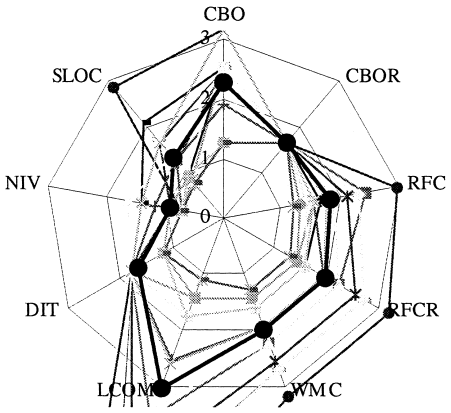

図 4 分類 CView のメトリクス値
Fig. 4 Metric values for CView classes.

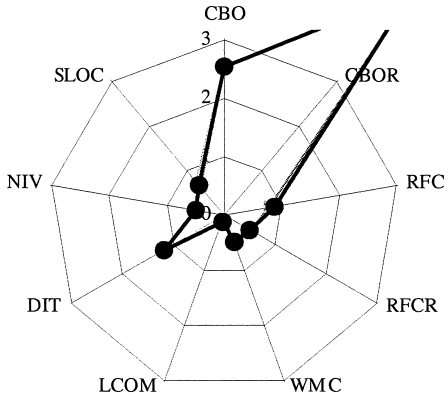

図 5 分類 CWinApp のメトリクス値
Fig. 5 Metric values for CWinApp classes.

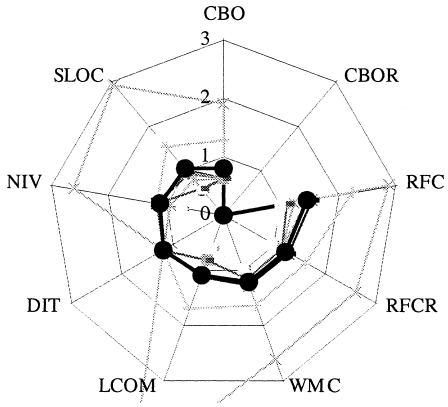

図 6 分類 CFrameWnd のメトリクス値
Fig. 6 Metric values for CFrameWnd classes.

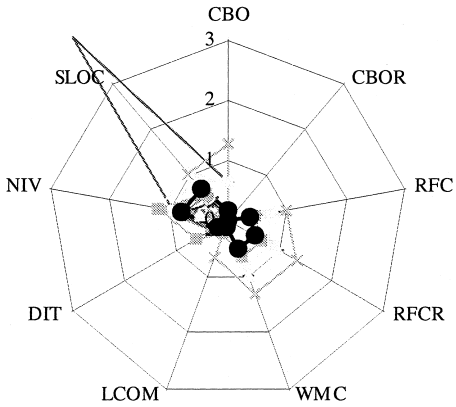

図 7 分類その他のメトリクス値
Fig. 7 Metric values for the other classes.

分類の平均値と，分類に属する個々のクラス（細い線）のメトリクスは互いに似た傾向を示しており，クラス分類が適切であったことの傍証となっている．

4.4 分　　析

本実験では，メトリクスの計測値からエラー修正時間を予測する式を用いて手法を評価した．メトリクスの計測値を独立変数，エラー修正時間を従属変数とす

表 9 全データによる予測式
Table 9 Estimation equations by all values.

Metric	All	CDialog	CDocument	CView	CWin App	CFrame Wnd	Others
(Const.)	-11.4	-22.1	-98.4	-22.9	0	-614	1.46
CBO							
CBOR							
CBON			47.9			70.6	63.3
RFC							
RFCR							
RFCN		-6.76				-67.8	-50.1
WMC						119	
LCOM		3.52	1.36			-21.5	
DIT							
NOC							
NIV				8.61			
SLOC	0.246	0.338				5.76	

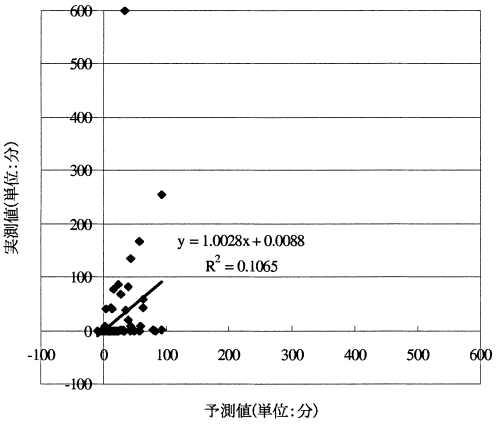

図 8 クラス分類を用いないエラー修正時間予測（全データ）
Fig. 8 Error fix time estimation by all data without classification.

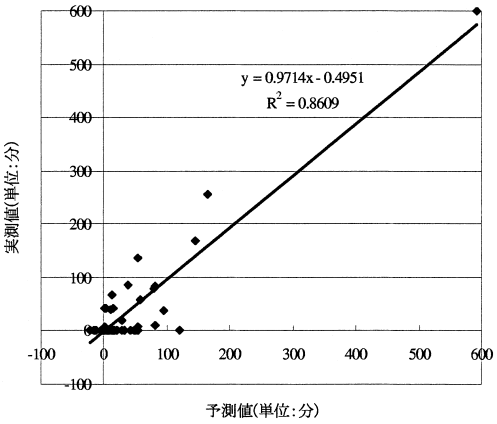

図 9 クラス分類を用いた修正時間予測（全データ）
Fig. 9 Error fix time estimation by all data with classification.

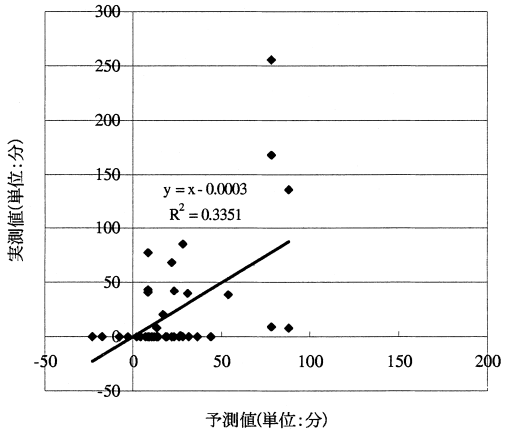

図 10 クラス分類を用いないエラー修正時間予測（外れ値を除く）
Fig. 10 Error fix time estimation execpt outliers without classification.

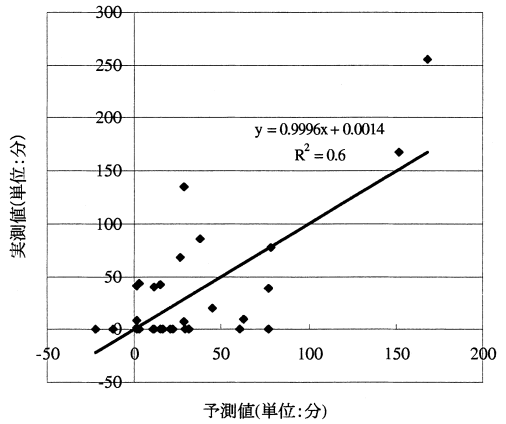

図 11 クラス分類を用いた修正時間予測（外れ値を除く）
Fig. 11 Error fix time estimation execpt outliers with classification.

る回帰式を，重回帰分析によって求めた．変数減少法を用い，サンプル数や寄与率に照らして統計的に有意とならない独立変数は取り除いてある．たとえば，CBON, CBOR，CBO の間には $CBON + CBOR = CBO$ という関係が成り立つため，3 変数がともに 1 つの回帰式に含まれることはない．

クラス分類ごとの回帰式の係数を**表 9** に示す．比較のために，分類をせずに求めた回帰式の係数も示した（All の欄）．エラー修正時間の予測値と実測値をプロットしたものを**図 8** および**図 9** に示す．クラス分類を行ったほうが予測精度が向上していることが分

表 10 外れ値を除いたデータによる予測式
Table 10 Estimation equations except outliers.

Metric	All	CDocument	CView	Others
(Const.)	2.89	-98.5	-23.0	1.46
CBO				
CBOR	63.3			
CBON		47.9		63.3
RFC				
RFCR				
RFCN				-50.1
WMC				
LCOM				
DIT	-8.64	1.37		
NOC				
NIV	4.91		8.64	
SLOC				

かる．

次に，突出したエラー（修正時間600分）が含まれているクラス分類（CFrame），少数しかエラーが発見されていないようなクラス分類（CDialog，CWinApp）を外れ値だと見なして，データから取り除いたうえで分析を行った．回帰式の係数を**表 10** に，エラー修正時間の予測値と実測値をプロットしたものを**図 10** および**図 11** に示す．この場合もやはりクラス分類を行ったほうが予測精度が向上している．

5. まとめ

本研究では，C++言語およびアプリケーションフレームワークを用いた開発において複雑度メトリクスを用いてエラー修正時間の予測を行う際に，クラス分類を行って予測精度を向上させる手法を提案し，実験によってその有効性を評価した．

今後の課題としては，

(1) クラス分類の精密化．Java 言語を用いて開発を行った場合には，クラスの継承以外にもインタフェースや匿名クラスといった，単なる継承とは見なせない機能がある．それらをクラス分類に用いることができるかもしれない．

(2) 分類クラス選出の自動化．本研究ではフレームワークに関する知識によって利用者が分類に用いられるクラスを選出する．クラス階層の構造やクラス間の関係，統計的手段を用いてクラス分類を自動化できれば，手法の利用がより簡単になると考えられる．

(3) より多くのプロジェクトに対してメトリクスの収集を行い，手法の有効性を評価する．

謝辞 評価実験にご協力いただいた，日本ユニシス株式会社の高橋優亮氏に感謝いたします．複雑度ツールの開発にご協力いただいた奈良先端科学技術大学院大学の高林修司氏（現 松下通信工業株式会社）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) Basili, V.R., Briand, L.C. and Melo, W.L.: A Validation of Object-Oriented Design Metrics as Quality Indicators, *IEEE Trans. Softw. Eng.*, Vol.20, No.22, pp.751–761 (1996).

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

4) Briand, L.C., et al.: Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems, *Proc. 9th International Symposium on Software Reliability Engineering*, pp.334–343 (1998).

5) Chidamber, S.R. and Kemerer, C.F.: A Metrics Suite for Object Oriented Design, *IEEE Trans. Softw. Eng.*, Vol.20, No.6, pp.476–493 (1994).

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) Kamiya, T., Kusumoto, S., Inoue, K. and Mohri, Y.: Empirical evaluation of reuse sensitiveness of complexity metrics, *Information and Software Technology*, 41, pp.297–305 (Apr. 1999).

9) Lorenz, M. and Kidd, J.: *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) Oman, P. and Pfleeger, S.L.: *Applying Software Metrics*, IEEE Computer Society Press (1997).

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

12) Pighin, M. and Zamolo, R.: A Predictive Metric Based On Discriminant Statistical Analysis, *Proc. 19th ICSE*, Boston, Massachusetts, USA, pp.262–270 (1997).

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

14) Cartwright, M. and Shepperd, M.J.: An Empirical Investigation of an Object-Oriented Software System, *IEEE Trans. Softw. Eng.* (1999, Accepted for publication). taken from http: // dec.bmth.ac.uk/ ESERG/ mshepperd/ OOMetrics.html
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 (1993).
16) UML Summary, ver. 1.1 (1997). taken from http://www.rational.com/

(平成 12 年 4 月 21 日受付)
(平成 13 年 3 月 9 日採録)

神谷 年洋（正会員）
平成 13 年大阪大学基礎工学研究科博士後期課程修了．科学技術振興事業団若手個人研究推進事業（さきがけ研究 21）［ポスドク活用型］「協調と制御」領域中小路グループグループメンバ．工学博士．オブジェクト指向ソフトウェアメトリクスおよび認知科学に関する研究に従事．

楠本 真二（正会員）
昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 8 年同大学講師．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．IEEE 会員．

井上 克郎（正会員）
昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学院博士課程修了．同年同大学基礎工学部情報工学科助手．昭和 59 年〜61 年ハワイ大学マノア校情報工学科助教授．平成元年大阪大学基礎工学部情報工学科講師．平成 3 年同学科助教授．平成 7 年同学科教授．工学博士．ソフトウェア工学の研究に従事．

毛利 幸雄
昭和 49 年青山学院大学理工学部物理学科卒業．同年日本ユニシス（株）入社．社内外に対する情報処理技術分野の教育に従事．現在総合教育部 IT 教育推進室に所属．ソフトウェア技術者協会会員．