オブジェクト指向開発におけるフォールト発生早期予測手法の一提案

神谷 年洋[†], 楠本 真二[†], 井上 克郎[†‡]

オブジェクト指向設計に対する複雑度メトリクスの中では, Chidamber と Kemerer らの提案したメトリクス(C&K メトリクス)が最もよく知られおり, 幾つかの評価実験も行われている. しかし, これらの実験において, C&K メトリクスは設計仕様書ではなく, ソースコードに適用されてきている. C&K メトリクスの一部は設計フェーズの後期にならないと入手できない情報に依存するからである. この論文では, 設計の早期フェーズにおいて C&K メトリクスを用いて, フォールトの発生するクラスを予測する手法を提案する. 具体的には, 先ず, 分析/設計/実装フェーズに 4 つのチェックポイントを導入し, 各チェックポイントで適用可能なメトリクスのみを用いてクラスの複雑さを計測する. 次に, 多変量ロジスティック回帰分析を行い, フォールトの発生を予測する.

On Early Phase Fault Estimation in Object-Oriented Development

Toshihiro Kamiya[†], Shinji Kusumoto[†] and Katsuro Inoue[†‡]

Chidamber and Kemerer's metrics are well-known object-oriented design complexity metrics. In the evaluations, their metrics were applied to source code because some of them depend on the information that can not be obtained until the end of design phase. This paper proposes a new method to estimate the fault-proneness of class by using C&K metrics at early design phase. In the proposed method, we introduce four checkpoints into the analysis/design/implementation phase, and use only the applicable metrics among the complexity metrics. Then we estimate the fault-proneness of the class by multivariate logistic analysis.

1. はじめに

ソフトウェアのテストは, 出荷するソフトウェアに含まれるフォールトを発見・除去するために必要な作業である. テストに費やされる労力は開発コストの50～80%に上るという報告もある[7]. 従って, テスト労力の削減は, ソフトウェア開発の生産性を改善する重要な手段となる. レビューはテスト労力を削減するためのもっとも効果的な手段の一つである. レビューやテストを効率よく行うためには, フォールトが含まれると予測されるモジュールを特定することにより, レビューやテストの労力をそのモジュールに集中させることが効果的である[1].

フォールトが含まれるモジュールを予測するために, 数々の複雑度メトリクスが提案されてきている. たとえば, Halstead のソフトウェアサイエンス[9]や, McCabe のサイクロマチック数[13]はよく知られたメトリクスであり, 多くのソフトウェア開発組織で用いられてきている. その後, Chidamber と Kemerer がオブジェクト指向ソフトウェア向けの 6 種類のメトリクスを提案した[5](以下 C&K メトリクス). これらのメトリクスはオブジェクト指向ソフトウェアが必然的に持つ主要な特徴を評価するメトリクスである.

Chidamber と Kemerer は 2 つのソフトウェア開発組織を観察し, オブジェクト指向プログラミング言語(C++と Smalltalk)で記述されたプログラムからメトリクス値を収集した. Basili らも, C& K メトリクスを用いてクラスにエラーが発生するかを予測する実験を行った[1]. しかし, これらの実験においては, C& K メトリクスはソースコードに適用されている. 彼らのメトリクスにはクラス内部の複雑度を計測するものがあり, そのような情報は, クラスのアルゴリズムと構造を決定してはじめて(すなわち設計フェーズの後期で), 入手可能になるからである. フォールトを修正するための労力を効果的に配分するためには, フォールトがどこに発生するかを早期のフェ

†大阪大学大学院基礎工学研究科
Graduate School of Engineering Science, Osaka University
‡奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

ーズで予測することが望まれる.

本研究では, 設計の初期段階において, オブジェクト指向ソフトウェア用のいくつかの複雑度メトリクスを用いて, クラスにフォールトが発生するかどうか(fault-prone か)を予測する新しい手法を提案する. 提案する手法では, OMT[14]に基づく分析/設計/実装フェーズに 4 つのチェックポイントを設ける. 各チェックポイントにおいて, 入手可能なプロダクトに関する情報(設計仕様書やソースコード)を用いて, 複雑度メトリクスのうち適用可能なもののみを適用する. 次に, 多変量ロジスティック回帰分析[19]を用いて, fault-prone なクラスを予測する. さらに, 提案する手法の有効性を評価するために, あるコンピュータ会社で行われた開発プロジェクトで実験を行った. 実験の結果, 提案する手法を用いて, 早期の段階でクラスのフォールト発生をある程度予測できることが確認された.

以下, 2.では, オブジェクト指向設計手法とオブジェクト指向複雑度メトリクスについて説明し, 3.では, OMT に基づいた分析/設計/実装フェーズに 4 つのチェックポイントを設けることによる, 新しいフォールト発生予測手法を提案する. 4.では, 提案した手法の評価実験について述べる. 5.でまとめと今後の課題を述べる.

2. 準備

2.1. オブジェクト指向設計手法

これまでに, 多くのオブジェクト指向設計手法が提案されてきている[2][6][14][15]. なかでも, Booch 法と Rumbaugh の OMT(Object Modeling Technique)がよく知られたオブジェクト設計手法である. OMT はオブジェクト指向の分析・設計の技術的な側面のほとんどを取り込んだ手法でり, 実際のプロジェクトの経験も取り入れたものになっている. 本研究では, オブジェクト指向設計手法としては OMT を対象とする.

OMT は, 分析, システム設計, オブジェクト設計の 2 つのフェーズから構成される[14]. 分析フェーズでは, アプリケーションとアプリケーションが動作するドメインを理解しモデル化する. 分析フェーズの出力はシステムの基本的な側面を捉えた次の 3 つの形式的なモデルである.

(1)オブジェクトモデル:オブジェクトとそれらの関係
(2)動的モデル:制御の動的なフロー
(3)機能モデル:データ加工の制約

システム設計フェーズでは, まず, システムの全体的なアーキテクチャを決定する. 次に, オブジェクトモデルを参照しながら, システムをサブシステムに分割する. このとき, オブジェクトを並列に実行できるタスクにグループ分けすることで, 並列性を取り出す. また, プロセス間の通信, データの格納, 動的モデルの実装に関する全体的な決定を行う. 設計のトレードオフに関する優先順位も確立する.

オブジェクト設計フェーズでは, 分析モデル(オブジェクトモデル, 動的モデル, 機能モデル)を洗練することで最適化を行い, 現実の設計を生成する. クラス内で用いられるアルゴリズムやデータ構造が決定される.

クラスの構造の観点からは, 以下の 6 つのステップで開発が進行する.

1. ターゲットシステムに含まれるクラスを特定する.
2. クラス間の参照関係を特定する.
3. クラスの属性を特定する.
4. クラス間の継承関係を特定する.
5. 機能モデルに基づいて操作を定義する.
6. 操作を実装するアルゴリズムを設計する.

2.2. 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プロダクトやプロセスを計測するための定量的な尺度である. ソフトウェアメトリクスは, プロダクトメトリクスとプロセスメトリクスの 2 種類に分類される[8]. 本研究ではプロダクトメトリクスのみを扱う. プロダクトメトリクスには, プロダクトの大きさ, 論理的な構造の複雑さ, データ構造の複雑さを, プロダクトの機能, それらを組み合わせたもの等がある. 特に, プログラムのソースコードの複雑度を計測するメトリクスは, 最も良く知られたメトリクスである. プロダクトの設計の善し悪し, 理解しやすさ, 修正の容易さを評価するために, 実装およびテストのフェーズで用いられる. たとえば, Halstead のソフトウェアサイエンス[9], McCabe のサイクロマチック数[13]は, 多くのソフトウェア開発組織で用いられてきている.

上記の複雑度メトリクスを用いて, オブジェクト指向言語で記述されたプログラムを分析することが可能である(たとえば, クラスの数を数える, メソッドを関数と見なして分析する). しかし, これらのメトリクスはオブジェクト指向プログラムの本質的な特徴を評価していない. その後, Chidamber と Kemerer はオブジェクト指向設計を対象とする 6 つのメトリクスを提案した[5]. C&K メトリクスは計測の理論に立脚したものであり, オブジェクト指向ソフトウェア開発者の経験的な洞察に一致する.

C&K メトリクスはクラスを対象としてその複雑さを評価するものであり, クラス間の関係やクラスの内部的な複雑さを計測する. 以下のように定義されている:

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_1 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[1]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た．この論文でも同じ仮定を用いるので，本稿では以下，この WMC を NIM(インスタンスメソッドの数;Number of Instance Methods)[12]と呼ぶ．

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木内での深さである．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．

C&K 以外のメトリクスとして，NIV(インスタンス変数の数;Number of Instance Variables)[12]を用いた．

NIV(インスタンス変数の数;Number of Instance Variables):

計測対象クラスのインスタンス変数の数である．

2.3. 複雑度メトリクスの評価

Chidamber と Kemerer は，実際の商用システムから C&K メトリクス値を計測し，メトリクスが実装に依存せず，設計の特徴を捉えることを実験的に評価した[5]．これらの結果は，実際にメトリクスを収集し，管理者がメトリクスを用いる可能性を示した．彼らはまた，メトリクスを Weyuker の性質[1]に照らして評価した．結論として，彼らのメトリクスは Weyuker の性質をおおむね満たす．Basili らは C&K メトリクスを実験的に評価し，従来のコードメトリクスの組み合わせよりも，フォールトの発生を予測するためのよい指標となることを示した[1]．

文献[1]と[5]の両方において，メトリクス値はソースコードから収集されている．その理由は次の 2 つである．(1)当時，オブジェクト指向設計仕様書の標準的な記述法が確立されていなかったため，実験のためだけに設計仕様書を作るのは不適当である．(2)ソースコードは設計仕様書を実装したものであり，メトリクス収集のために必要な情報はソースコードに含まれる．

設計仕様書が生成されるフェーズは，ソースコードが実装されるフェーズよりも早期である．従って，メトリクスを設計仕様書に適用することで，より効果的にフォールト発生の予測を用いることが望ましい．フォールト発生の予測に基づいて，資源割り当てやスケジューリングを行うことで，フォールトの効果的な検出を行うことが期待される．

しかし，C&K メトリクスのすべてを設計仕様書に適用するのは非常に困難である．たとえば，RFC と LCOM の計算をするためには，メソッド内部で用いられているアルゴリズムやメソッド間の呼び出し関係といったクラス内部の詳細な情報が必要である．これらの情報が記述されるのは，通常，設計フェーズの後期，実装フェーズの直前である．3.では，この問題の解決策を示し，設計仕様書にメトリクスを適用する具体的な手法を提案する．

3. 提案する手法

3.1. 基本方針

本研究においては，分析/設計/実装フェーズを，進行するにつれてプロダクトに関する知識が増大していく一連のプロセスであると捉える．2.で述べたように，OMT に基づく分析・設計・実装のフェーズにおいて，メ

[1] Weyuker はソフトウェアメトリクスが満たすべき一連の必要条件を考案し，既存のメトリクスをこの必要条件に照らして評価した[17]．

トリクスの中には設計フェーズの早い段階で適用可能なものと，設計フェーズの遅い段階，実装フェーズの直前でしか適用できないものがある．この事実に基づき，設計フェーズの各段階で，設計仕様書に対して適用可能なメトリクスを用いてフォールト発生の予測を行う．

まず，計測の観点から開発プロセスに 4 つのチェックポイントを導入し，各チェックポイントでどのような情報が設計仕様書に付け加わるかを明らかにする．次に，各チェックポイントで開発される設計仕様書で適用可能なメトリクスの部分集合を定義する．最後に，各チェックポイントにおいて，フォールト発生を，適用可能なメトリクスの多変量ロジスティック回帰分析を用いて予測する．

3.2. チェックポイントと適用可能なメトリクス

2.では，OMT の分析・設計フェーズを，クラスの構造を詳細化していく過程と見立て，6 つのステップに分割した．この分割に基づいて，設計仕様書にメトリクスを適用するために，分析/設計/実装フェーズに 4 つのチェックポイントを設ける．

(CP1)実体と関係(entity and relation)

CP1 はステップ 1, 2, 3 が完了した時点である．クラス間の参照関係とクラスの属性が決定されている．参照関係はクラス間の結合(coupling)に対応し，属性はインスタンス変数に対応する．CP1 においては，導出関係は決定されておらず，クラスライブラリ中のどのクラスが再利用されるかも設計仕様書には記述されていない．他方で，NIV は属性の情報から計算される．CBO は参照関係から計算されるが，クラスライブラリ中の再利用されるクラスへの参照は明確に記述されていないので，CBO の値は正確ではない．

(CP2)構造と継承

CP2 はステップ 4 と 5 が完了した時点である．すなわち，クラスの導出関係とクラスのメソッドが決定されている．導出関係を決定するために，クラスの継承木が明確に記述される．従って，DIT が導出関係から計算される．NIM はメソッドに関する情報から計算できる．再利用されるクラスが決定されているので，CBO は正しく計算される．

(CP3)アルゴリズム

CP3 はステップ 6 が完了した時点である．すなわち，各メソッドのアルゴリズムとメソッド間の呼び出し関係が決定されている．この情報によって，LCOM と RFC が計算される．

(CP4)実装

CP4 はソースコードが実装された時点である．各クラスについて，SLOC(ソースの行数)が計算される．

CBO は CP1 と CP2 で計算される．CBO は対象となるクラスとそれ以外のクラスとの結合の数を数えるメトリクスである．CP1 の CBO は計測対象クラスと新規に開発されたクラスとの結合のみを数えるので，CBO の値は C&K メトリクスのオリジナルの定義に従って計算されるわけではない．しかしながら，過去の研究によって，CP1 の時点の CBO はフォールト発生と高い相関を持つことが示されている[10]．それゆえ，以下の 2 つのメトリクスを導入してこれらの CBO を区別する．

CBON(Coupling Between Object classes Newly-developed)：

CBON は対象となるクラスと，新規に開発されたクラスとの結合を数える．

CBOR(Coupling Between Object classes Reused)：

CBOR は対象となるクラスと，再利用されたクラスとの結合を数える．

定義より，CBO は CBON と CBOR の合計になる．

チェックポイントと，チェックポイントにおいて計算可

表 1 チェックポイントと適用可能なメトリクス

Table 1 Checkpoint and available metrics

チェックポイント	増加する情報	適用可能なメトリクス
(CP1)実体と関係	クラス間の関係，クラスの属性	NIV, CBON
(CP2)構造と継承	クラスの継承構造，メソッド，再利用されるライブラリ	NIV, CBON, CBOR, CBO, NIM, DIT, NOC
(CP3)アルゴリズム	メソッドのアルゴリズム	NIV, CBON, CBOR, CBO, NIM, DIT, NOC, RFC, LCOM
(CP4)実装	ソースコード	NIV, CBON, CBOR, CBO, NIM, DIT, NOC, RFC, LCOM, SLOC

能なメトリクスをまとめたものを表 1 に示す.

3.3. 多変量ロジスティック回帰分析

文献[1][3]において, 多変量ロジスティック回帰分析がプログラムのフォールト発生を予測するのに用いられている. 本研究でもこの手法を用いる. メトリクスの fault-prone 予測性能(対象のクラスにフォールトが発生するかを予測する性能)を評価するためには, まず, クラスのメトリクス値を入力とし, 真偽値(予測)を出力する関数を作る必要がある. 出力の真偽値はクラスがフォールトを持つか持たないかを示す. 関数に観測されたメトリクス値を与えて予測を行う. その後, 予測モデルと実際に観測されたフォールトを比較することで, モデルの正確さを判定し, メトリクスの性能を評価する.

多変量ロジスティック回帰モデルは以下の関係式を用いる.

$$P(X_1, ..., X_n) = 1 / (1 + \exp(-(C_0 + C_1 \cdot X_1 + ... + C_n \cdot X_n)))$$

ここで, P は与えられたクラスにエラーが見つかる確率であり, X_i はクラスのメトリクス値である. もし与えられたメトリクス値が P を 0.5 以上にするなら, クラスはフォールトを持つ(fault-prone である)と予測する. この式において, $Z = C_0 + C_1 \cdot X_1 + ... + C_n \cdot X_n$ とおけば, $P(Z) = 1 / (1 + \exp(-Z))$ となる. この P と Z の関係は S 字カーブ(図 1 参照)になる. このような S 字カーブは X_i と P の関係が単調である限り, 2 値の分類に適用可能である.

係数 C_i を決定する際には, 最尤度(maximum-likelihood)基準が用いられる. すなわち, 係数は観測された結果をもっともよく反映するような値が選ばれる. メトリクス(変数)$X_1 ... X_n$ の相関が強い(独立性が低い)場合, 冗長な変数が含まれると, 導かれる係数を不適切あるいは誤解を招くようなものにしてしまう. このような, 意味がないあるいは有害な変数は, 段階的変数選択によって取り除かれる. 数種類の選択アルゴリズム存在する. この論文では変数増加法と減少法の両方を変数選択アルゴリズムとして用いた.

4. 実験的評価

4.1. 実験の概要

実験データは, 1997 年 8 月にある企業の新人研修で行われた C++プログラム開発演習から収集された. 演習の概要は以下の通りである:

(1)開発者は会社の新入社員であり, 大学あるいは大学院を卒業し, 1997 年 4 月に入社した. 事前に行われた講義と演習により, オブジェクト指向設計と C++言語によるプログラミングを修得している.

(2)16 の開発チームが, 同一の要求仕様書に基づいてメール配送システムを作成する. このシステムは分散ネットワーク環境で動作し, ASCII エンコードされたメールを送受信する. 開発開始時点で, 要求仕様書, サブシステム(それぞれ SMTP サーバー, POP サーバー, DELIVER サーバー, SMTP クライアント, POP クライアント)への分割, サブシステムのインターフェイス設計がチームに与えられる. それぞれのチームのリーダーが, チームのメンバーに開発すべきサブシステムを割り当てる.

(3)チームは 4 から 5 人の開発者で構成される. インストラクターが, 開発者の能力を考慮して, 開発者をチームに編成する.

(4)チームがプログラムの完成を通知すると, インストラクターが受け入れテストを行う.

(5)プログラムは C++で実装される. 開発環境は Visual C++であり, 開発には Microsoft Foundation Class(MFC)がアプリケーションフレームワークとして用いられる. ユーザーインターフェイスとソケットサービスが MFC のクラスから派生したクラスとして実装された.

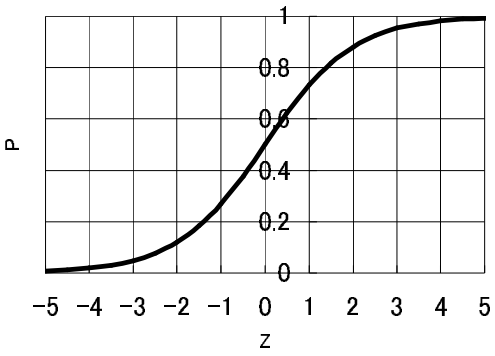

図 1 P と Z の関係
Figure 1 Plot of P with Z

4.2. 実験データ

開発者ごとに, メトリクスとフォールトデータを収集した. 本実験では OMT による設計仕様書は収集できなかった. そこで, ソースコードは設計仕様書を実装したものであるから, 設計仕様書のすべての情報を含んで

いるという仮定に基づいて，各チェックポイントにおけるメトリクスの値をソースコードから収集した計測値で代用した．開発者は各々割り当てられた PC 上で作業を行い，ネットワーク経由でサーバーが 1 時間ごとに，ソースコードを収集した．メトリクス値の算出には，C++プログラムから 9 種のメトリクスを抽出するツール[11]を用いた．本実験では開発作業を記録するためのツールも準備され，フォールトデータの収集に用いられた．収集されたフォールトデータは，(1)コードレビューのフェーズとテストフェーズで発見されたフォールト，(2)これらのフォールトを修正するために変更されたクラス，(3)フォールトを修正するために費やされた労力(時間)，である．

フォールトデータを残さなかった，あるいはデータが欠落している開発者は，分析の対象から外した．結果として，17 人のデータ(141 個のクラス，80 個のフォールト)が分析対象になった．

表 2 はメトリクスの統計量である．この結果は，クラスが比較的小規模なものであったことを意味する．NIV と NIM がともに 0 のクラスがあったが，このクラスは実装のすべてを親クラスから継承していた．

4.3. 分析

表 3 に，多変量ロジスティック回帰分析によって算出された予測モデルの係数を示す．CBO，CBOR，CBON には依存関係があるため(*CBO = CBOR + CBON*)，3 つがともに予測式に含まれることはない．

表 2　141 の C++クラスの統計量

Table 2 Descriptive statistics of the 141 C++ classes

メトリクス	最小	最大	中央	平均	標準偏差
NIV	0	14	3	4.00	2.67
CBO	0	5	1	1.39	1.59
CBON	0	3	0	0.53	0.99
CBOR	0	4	1	0.86	0.99
NIM	0	22	3	5.73	4.86
DIT	0	6	4	3.44	1.41
NOC	0	0	0	0.00	0.00
RFC	0	27	7	8.23	6.81
LCOM	0	190	3	22.42	36.84
SLOC	0	420	71	96.43	81.01
エラー数	0	17	0	0.57	1.93
Et(分)	0	599	0	12.68	58.94

Et はエラーの修正に要した時間

DIT は複雑さに対する負の要因となった．この原因は，本実験では多くの「ダイアログ」クラスが作られたが，機能が単純であったにも関わらず比較的大きな DIT を持ったことである(ダイアログクラスの DIT 値は 4 であった)．観測された NOC はすべて 0 であった(表 2 参照)，そのため NOC は正しく予測式から取り除かれている．LCOM は CP4 において予測式からとりぞかれているが，これは[1]の結果と合致する．

表 4, 5, 6, 7 は各チェックポイントで収集されたデータを多変量ロジスティック回帰分析することで得られた予測モデルを示している．たとえば，表 4 では，112 個のクラスがフォールトを持たないと予測され，実際にフォールトが発見されなかった．2 個のクラスはフォールトがあると予測され，実際にはフォールトが発見されなかった．18 個のクラスはフォールトを持たないと予測されたが，実際にはフォールトが発見された(43 個のフォールトを含んでいた)．9 個のクラスはフォールトを持つと予測され，実際にフォールトが発見された(37 個のフォールトを含んでいた)．

ここで，予測式の精度を評価するために，3 つの指標を導入する:

正確性(Correctness): 正しくフォールトがあると予測されたクラスの割合(%)．

完全性(Completeness): フォールトがあるクラスが検出された割合(%)．

フォールトベースの完全性: フォールトがあると予測されたクラスで実際に検出されたフォールトの割合(%)．

表 3　各チェックポイントにおける係数

Table 3 Coefficients at Each Checkpoint

メトリクス	係数			
	CP1	CP2	CP3	CP4
定数 C_0	-3.37	-1.23	-1.31	-2.69
NIV	0.420	EL	EL	EL
CBON	EL	EL	EL	EL
CBOR	-	0.934	0.890	EL
CBO	-	EL	EL	EL
NIM	-	0.336	EL	EL
DIT	-	-1.16	-1.28	-0.663
NOC	-	-	EL	EL
RFC	-	-	0.284	EL
LCOM	-	-	-	EL
SLOC	-	-	-	0.0302

「EL」はそのメトリクスが変数減少法によって予測式から取り除かれたことを示す．「-」はそのメトリクスがそのチェックポイントでは適用できないことを示す．

表 4 CP1 におけるフォールト予測

Table 4 Fault Prediction at CP1

予測		フォールト無	フォールト有
実測	フォールト無	112	2
	フォールト有	18(43)	9(37)

括弧の外の数字はクラスの数．括弧内の数字はクラスで発見されたフォールトの数

表 5 CP2 におけるフォールト予測

Table 5 Fault Prediction at CP2

予測		フォールト無	フォールト有
実測	フォールト無	109	5
	フォールト有	11(20)	16(60)

表 6 CP3 におけるフォールト予測

Table 6 Fault Prediction at CP3

予測		フォールト無	フォールト有
実測	フォールト無	111	3
	フォールト有	9(18)	18(62)

表 7 CP4 におけるフォールト予測

Table 7 Fault Prediction at CP4

予測		フォールト無	フォールト有
実測	フォールト無	111	3
	フォールト有	8(14)	19(66)

これらの基準は以下のように定義される．

$$Correctness = C_{PFAF} / (C_{PFAF} + C_{PFAN})$$
$$Completeness = C_{PFAF} / (C_{PFAF} + C_{PNAF})$$
$$Completeness_{faultbased} = E_{PFAF} / (E_{PFAF} + E_{PNAF})$$

ここで，C_{PFAF} はフォールトがあると予測され実際にフォールトがあったクラスの数，C_{PFAN} はフォールトがあると予測されたが実際にはフォールトがなかったクラスの数，C_{PNAF} はフォールトがないと予測されたが実際にはフォールトがあったクラスの数，E_i は対応する C_i のクラスで発見されたフォールトの数である．

チェックポイント CP1 から CP4 での fault-prone 予測精度を表 8 に示す．後期のチェックポイントほど，より正しく予測を行える．CP4 は開発プロセスの最終フェーズであり，本実験における予測精度の上限である．

CP1 においては，完全性は低く(33%)，正確性は高い(82%)．よって，フォールトが発生しそうなクラスを「シード」する目的で CP1 における予測を用いることができる．シードされたクラスは重点的にレビューされテストされるクラスの候補になる．また，シードされたクラスの分布が設計レビューの判断基準になる．たとえば，シードされたクラスが設計仕様書の重要な部分に集中していて，かつ，テストが困難な部分であるなら，再設計を行うということが考えられる．

表 8 フォールト予測の精度

Table 8 Fault Predict Precision at Checkpoint

チェックポイント	CP1	CP2	CP3	CP4
正確性(%)	82	76	86	86
完全性(%)	33	59	67	70
フォールトベースの完全性(%)	46	75	78	83

CP2 では，C&K メトリクスのメソッドのアルゴリズムに関するものは用いられていないにも関わらず，CP4 を予測精度の上限と比較して，かなりよい予測精度となっている(「完全性」ではほかのチェックポイントよりも低くなっているが，「フォールトベースの完全性」ではよい成績を収めているので，フォールトを予測するという当初の目的に照らせば問題はないと考えられる)．この結果は，設計フェーズにおいて，アルゴリズムが決定していない段階で(当然ソースコードも用いず)，設計仕様書からエラーの発生を予測する可能性を示唆している．

CP3 での予測は CP2 での予測に比べて，予測精度がそれほど向上していない． 我々は，CP3 における予測精度は，「細粒度」C++設計メトリクス[3]を援用することで改善できると考えている．Chidamber らも，WMC の値は，計測されるメソッドの実装に依存すると述べている．たとえば，サイクロマチック数等を用いてメソッドの複雑さを適正に重み付けする WMC を用いることで，予測精度は改善されると考えられる．

5. まとめ

本論文では，オブジェクト指向複雑度メトリクスを用いて，設計フェーズの早期にフォールトの発生を予測する方法を提案した．この手法では，分析/設計/実装フェーズに 4 つのチェックポイントを導入した．これらのチェックポイントでは，特定の部分メトリクスのみが計測可能であった．さらに，この手法を実験的なプロジェクトに対して適用した．分析結果は提案する手法の評価と有効性を示している．

今後の課題として，次の(1)～(3)を考えている．

(1)提案する手法の拡張

今回利用したメトリクス以外のメトリクスに対して，提案した手法を用いることで，より正確な予測を行う．

(2)動的な複雑度メトリクスの開発

C&K メトリクスはオブジェクト指向ソフトウェアの静的な複雑さを評価する．しかし，オブジェクト指向設計仕様書は動的な情報も含む(たとえば，UML[18]の状態遷移図，シーケンス図，コラボレーション図)．そのような動的な複雑さを評価する方法が必要である．

(3)設計仕様書に対するメトリクスツールの開発

現在，UML(Unified Modeling Language)[18]がオブジェクト指向設計仕様書の標準記述言語として提案されている．本研究では，設計仕様書に対するメトリクスの計測値の近似としてソースコードに対する計測値を用いた．筆者らは，実際に設計仕様書から計測する実験を行ないたいと考えている．UML で記述された設計仕様書に対する計測ツールを開発中である[16]．

謝辞

4.の評価実験で協力いただいた，日本ユニシス株式会社の毛利幸雄氏ならびに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．本研究を進めるに当たり，一部，文部省科学研究費補助金 奨励研究(A)(課題番号 10780191)の補助を受けた．

参考文献

[1] V. R. Basili, L. C. Briand, 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] G. Booch: *Object Oriented Analysis and Design with Applications*, The Benjamin / Cummings (1994).

[3] L. C. Briand, P. Devanbu, and W. Melo: “An Investigation into Coupling Measures for C++”, *Proc. of the19th Int'l Conference on Software Eng.*, Boston, USA, pp.412-421 (1997).

[4] L. C. Briand, J. Daly, V. Porter, and J. Wust: “Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems”, *Proc. of the 9th Int'l Symposium on Software Reliability Eng.*, Paderborn, Germany, pp.334-343 (1998).

[5] 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).

[6] P. Coad and E. Yourdon: *Object Oriented Analysis, 2nd ed.*, Yourdon Press (1991).

[7] J. S. Collofello and S. N. Woodfield: “Evaluating the effectiveness of reliability-assurance techniques”, *Journal of Systems & Software*, Vol.9, No.3, pp.191-195 (1989).

[8] N. E. Fenton, and S. L. Pfleeger, *Software Metrics, A Rigorous & Practical Approach. 2nd ed.*, Thomson, London (1996).

[9] M. H. Halstead: *Element of software science*, New York, Elsevier North-Holland (1977).

[10] 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, Vol. 117-C, No.11, pp.1586-1592 (1997).

[11] 神谷，高林，楠本，井上: “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”，オブジェクト指向最前線’98 情報処理学会 OO’98 シンポジウム, pp. 37-44 (1998).

[12] M. Lorenz and J. Kidd: *Object-Oriented software metrics*, New Jersey, Prentice Hall (1994).

[13] T. J. McCabe: “A complexity measure”, *IEEE Trans. on Software Eng.*, Vol. SE-2, No.4, pp.308-320 (1976).

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

[15] S. Schlaer and S. Mellor: *Object Oriented System Analysis*, Prentice Hall (1988).

[16] 上村，柏本，楠本，井上: “UMLで記述された設計仕様書からのファンクションポイント計測手法”，1999 年電子情報通信学会総合大会 (発表予定).

[17] E. J. Weyuker: “Evaluating software complexity measures”, *IEEE Trans. on Software Eng.*, Vol.14, No.9, pp.1357-1365 (1988).

[18] *UML Modeling Language, Standard Software Notation*. Available at <http://www.rational.com/>.

[19] *SPSS Base 8.0 Applications Guide/Professional Statistics*, SPSS (1998).