社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2005-68(2005-12)

保守の影響波及範囲に基づいたレガシーシステムの障害予測

小林 健一† 吉野 利明† 井上 克郎‡ 早瀬 康裕‡ 松尾 昭彦† 上村 学†

†(株) 富士通研究所 ITコア研究所 ソフトウェアイノベーション研究部
‡大阪大学大学院情報科学研究科コンピュータサイエンス専攻
E-mail: † {kenichi, yoshino.toshi, a_matsuo, kamimura.manabu}@jp.fujitsu.com,
‡ {inoue, y-hayase}@ist.osaka-u.ac.jp

あらまし ソフトウェア保守において、モジュールやサブシステムのリスクを予測するメトリクス「インパクトスケール」を保守の影響範囲の広さを基に定義する。実稼動COBOLアプリケーションから収集された障害発生履歴をもとに、インパクトスケールが障害発生率へ及ぼす寄与を既存のメトリクスと比較して評価する。

キーワード ソフトウェア保守、メトリクス、障害予測、レガシーシステム

Predicting fault-proneness of legacy systems based on change impact

Kenichi KOBAYASHI†, Toshiaki YOSHINO†, Katsuro INOUE‡, Yasuhiro HAYASE‡,
Akihiko MATSUO† and Manabu KAMIMURA†

† Fujitsu Laboratories Limited
‡ Graduate School of Information Science and Technology, Osaka University
E-mail: † {kenichi, yoshino.toshi, a_matsuo, kamimura.manabu}@jp.fujitsu.com,
‡ {inoue, y-hayase}@ist.osaka-u.ac.jp

Abstract In software maintenance, we define a new metrics "Impact Scale" which is based on the size of the change impact in maintenance activities. We evaluate the Impact Scale how effective it is in predicting faults by using fault reports collected from COBOL applications running in the real business. We also compare the results with existing software metrics.

Keywords software maintenance, metrics, fault prediction, legacy system

1. はじめに

長年に渡って保守・運用されているレガシーソフトウェアの保守コストは、年とともに増加する一方である。近年は保守継続か再構築か、または保守のアウトソース化かといった戦略的な意思決定が常に要求されているにも関わらず、ソフトウェアの現状を適切に把握する手段が不足している。また、アウトソース化を請け負うベンダにとっても、巨大なソフトウェアの保守を新規に請け負う際にはそのリスクの評価が最も大きな関心事である。

そこで我々は、ソフトウェアの現状を把握するためのメトリクスとして、ソフトウェアの構造的な複雑度を定量化するためのメトリクス「インパクトスケール」を提案する。これは、保守においては、経年の劣化度合いを把握して構造的な問題の発生を察知する道具として、そして根本的な保守戦略の見直しのための判断材料として役立てることを目指す。また、新規保守の請負時においては、構造的な劣化をそのソフトウェアの保守時のリスクとみなし、請負の可否や請負額の判断ツールとして活用することを目指す。

2. インパクトスケール

2.1. 保守の困難さ

本稿では保守の困難さを保守のための変更の難易度と捉えて、次の3つの要素に分解する。ただし、これらの要素にはサイズの成分を含めず、単位量当たりの難しさとして扱えるようにする。

(A) プログラム内の複雑度
(B) プログラム・データ間の構造的な複雑度 (=アーキテクチャ複雑度)
(C) その他の要因からなる複雑度

ここで、プログラムとはソフトウェアの適度に分割された実行処理単位とする。例えば、COBOL 言語であれば 1 本のプログラムはそのまま用い、Java 言語などのオブジェクト指向言語では 1 つのクラスを 1 つのプログラムとみなす。

(A)と(B)はソースコードから得られる情報から計算できるものが望ましい。これは、レガシーソフトウェアの保守という環境では多くの情報が整備されていない、または欠落しているため、確実に頼れる情報源がソースコードのみであることが多い、という制約から来る要請である。ただし、他の情報源を補助的に利用するのは構わない。

その他の要因、例えば、人的スキル、計算機環境の変化、ビジネスニーズの変化などはすべて(C)に含まれるとする。本稿では(A)と(B)をモデル化し、(C)はそのモデルでは扱わない誤差量とする。

これらの複雑度は、ソフトウェアが開発された時点では比較的低い値に抑えられている(そのような設計が良い設計とされる)。しかし、保守による変更を繰り返すたびに、当初の設計の意図を外れて、または止むを得ず、複雑度は増加することになる。

(A)のメトリクスとしては、McCabe の循環的複雑度[1](以降 Mc と称す)に基づいて決める。ただし Mc は対象コードが大きくなるほど大きくなるメトリクスなので、サイズ要素を除く加工が必要である。一方、(B)のメトリクスとして、我々は「あるプログラムに変更を施したとき、その変更が波及する範囲の大きさ」をそのプログラムに関するアーキテクチャ複雑度として定義することにした。この新しいメトリクスを「インパクトスケール」と呼ぶ。

2.2. インパクトスケールの定義

対象とするソフトウェア P を無向グラフ G_P=<V, E> で表す。V の要素である頂点 v は P に属するプログラムまたはデータエンティティである。E の要素である辺 e=(u, v)は頂点 u, v 間に次の影響波及関係の(a),(b),(c)または

(a) プログラム u はプログラム v を呼び出す
(b) プログラム u はプログラム v から呼び出される
(c) プログラム u はデータエンティティ v にアクセスする

次に、関数 S(v, d)を「グラフ G_P 上で、頂点 v から到達可能な頂点のうち、最短距離が d である頂点の数」と定義する。また次の関数も定義する。

$$is(v,d,\alpha)=\sum_{i=1}^{d}\alpha^{i-1}S(v,i)$$

ここで、頂点 v のインパクトスケール IS(v)を

$$IS(v)=is(v,d_0,\alpha_0)$$

で定義する。d_0 と α_0 は定数であり、ここでは仮に d_0=10、α_0=0.5 とする。関数 S(v, d)に頂点 v の代わりに頂点の集合 M を入れた場合、関数 S(M, d)は以下の様に、要素の値の平均値として定義する。

$$S(M,d)=\frac{1}{|M|}\sum_{v\in M}S(v,d)$$

関数 is(), IS()に集合 M を適用する場合についても同様である。

インパクトスケールの定義の意味づけは、最初に述べたとおり、「プログラム v への変更の影響が波及する大きさ」を反映している。α は 0～1 の範囲の値をとり、距離が離れるほど影響が及ぶ確率が下がるという事象をモデル化している。

図 1(A)はあるアプリケーションから生成したグラフ[1]である。プログラム b のインパクトスケールは $IS(b)=2\cdot 1+1\cdot 0.5+1\cdot 0.5^2=2.75$ である。一方、(B)は図 1(A)にプログラム e からプログラム b への呼出しを行う修正を加えた。この場合、プログラム b のインパクトスケールは IS(b)=3・1+1・0.5=3.5 となる。(B)はアプリケーションが複雑になり、インパクトスケールが上昇している。

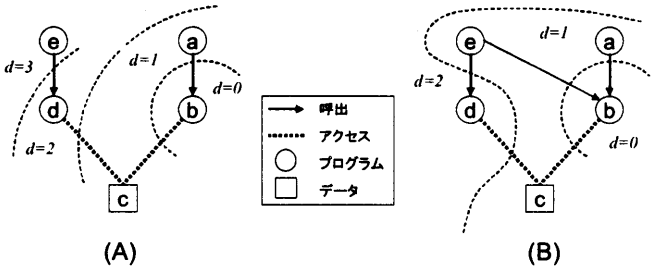

図 1 インパクトスケールの計算例

2.3 計測手順

ソフトウェア P に対し、以下の手順でグラフ G_P を得る

1. プログラムの単位を定義
2. プログラムをリストアップ
3. データエンティティの単位を定義
4. データエンティティを抽出
5. プログラム間の呼出関係を抽出
6. データアクセスを抽出
7. データアクセスの種類(「参照のみ=R」か、「更新=W」か)を判定。不明の場合は W とみなす

[1]図では呼出し関係が矢印で表示されているが、あくまでも無向グラフである

インパクトスケールの計測のためには、厳密な静的解析の必要は無い。間接呼出や間接参照を完全に抽出するためには、ソース以外の情報（業務データや実行トレースなど）を必要とするが、これらは多くの場合入手が難しい情報である。入手できない場合、単に無視すればよい。情報の欠落により蒙る不利益は計測精度の低下である。計測コストと計測精度の関係はトレードオフであり、どこでバランスさせるかは、対象ソフトウェアの診断者が自らの目的に応じて決定すればよい。

しかし、ソフトウェアPに、ディスパッチャやアクセッサなどのハブとなるプログラムがある場合、例えばグラフが図 2のようなときは、多くの場合、(d)→(アクセッサ)→(a)や(d)→(アクセッサ)→(b)という関係はあるが、(a)→(アクセッサ)→(b)という関係は存在しない。だが、単純な呼出関係の解析では過剰に関係を抽出してしまい、偽の関係まで抽出してしまう。この例ではアクセッサを利用するすべてのプログラムの間に関係が生じてしまうことになる。これではプログラムを特徴付ける指標としての役を成さない。そこで、グラフ G_P の経路を辿る際に、次のような直前の辺の状態によって探索を打ち切るカットルールを付加する。

I. 前の辺が呼出し方向で、今の辺が呼出される方向のみの場合、探索を打ち切る
II. 前の辺が呼出される方向で、今の辺が呼出し方向のみの場合、探索を打ち切る
III. 前の辺が参照のみのアクセスで、今の辺がアクセスの場合、探索を打ち切る

これにより、偽の影響波及を擬似的に抑制する。

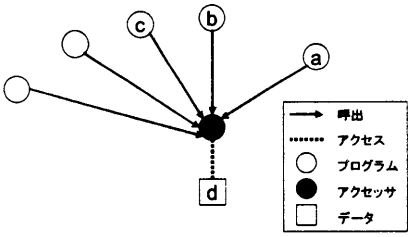

図 2 アクセッサによる偽の影響波及

2.4 計算量

インパクトスケールの値は、測定対象の全プログラムにと 1 から d_0 までの距離に関して、関数 S(v, d) の値を計算すれば求められる。

しかし、前節で導入したカットルールのために、無向グラフ G_P における関数 S の計算は単純な探索では実現できない。しかし、本稿では手順を割愛するがグラフの変換により、グラフ G_P 上のカットルール付き探索を、有向グラフ上の幅優先探索に帰着させることができる。この変換された有向グラフ G_P' の頂点数は 2|V| である。最終的な計算量は O(|V|(|V|+|E|))となる。通常は|E|は|V|と同オーダーと期待できるから実用的には $O(|V|^2)$である。

3. 実験

現在実稼動している COBOL アプリケーションにて、インパクトスケールの測定を行う。そのインパクトスケールと保守の困難さとの関連を調べるために、インパクトスケールを利用してプログラム毎のフォールト率を予測するモデルを作り、実績値と比べることにより、インパクトスケールの効果を評価する。

3.1 測定環境

実験の対象となるアプリケーション P の特徴を以下に記す。

- 全体規模: 約 5 百万行
- モデル構築対象：
  - バッチファイル 3451 本
  - COBOL プログラム 7295 本
  - データベース 206 テーブル (抽出分のみ)

3.2 波及範囲の測定

本実験では、プログラムの単位を、1 本のバッチファイル(以降、BATCH)またはプログラム(以降、PG)とし、データエンティティの単位をデータベース上のテーブル(以降、TABLE)とする。

BATCH と PG の呼出関係を解析した結果、BATCH→PG の呼出は 14447 回、PG→PG の呼出は 29125 回あった。

簡単なプログラム解析にて、TABLE を 206 エントリ抽出した。PG→TABLE のアクセスは 2854 回あった。この解析では R と W の判別は行っていないため、すべて W として扱う。

上記からなるグラフ G_P=<V, E>を構築した。すべての距離について S(V, d) を計算した。|V|=10952, |E|=43426 に対し、計算時間は C++言語で記述された計測プログラムを Xeon(2.4GHz)の PC で実行し、11.4 秒であった。最大経路長(d の最大値)は 24 であった。図 3 に求めた S(V,d)を示す。

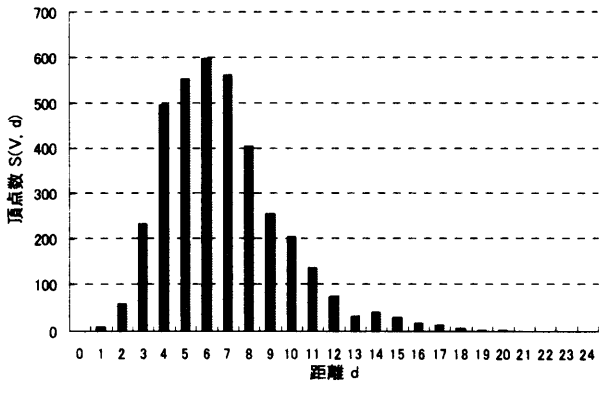

図 3 距離と到達頂点数

3.3 フォールト予測モデル

PG毎のフォールト率を予測するモデルを立て、そのモデル内のインパクトスケールの寄与を調べる。PG7295本のうち、未使用のものと、障害情報採取開始時期に作成されていなかったものを除いた6627本が標本である。Mを標本の集合とする。このうち、障害情報採取期間40ヶ月の間に238本に障害が発生した。

他変数から、障害発生の有無のような2値の結果を予測するモデルとしては、多重ロジスティック回帰分析がよく使われており[2][3][4]、ここでも採用する。これはパラメータを $\mathbf{X}=(X_1,X_2,...)$、目標値をYとしたときに、

$$Y=\begin{cases}1 & ;(Y^*>0.5)\\ 0 & ;(Y^*\le 0.5)\end{cases}$$

$$Y^*=\frac{1}{1+e^{-Z}}$$

$$Z=\beta_0+\beta_1X_1+\beta_2X_2+...$$

とするモデルである。実際にフォールト率を与えるのは Y^*である。この式が既定の閾値(上の式では0.5)を越えれば、フォールトと判定する。

モデルに入力するパラメータ $\mathbf{X}$ は、測定したメトリクスの中から、予測精度が最も高くなるように選ぶ。選択基準として、AIC(Akaike's Information Criteria)[5]を用いた。モデルの係数 β_0, β_1, β_2, ... は最尤推定法にて決定する。

モデルのパラメータの選択のために、4つのメトリクスのカテゴリ(サイズ、時間、プログラム内の複雑度、アーキテクチャ複雑度)を以下のように設定した。

まず、障害発生率はソースコードのサイズと相関が強いことは自然に予想できる。サイズカテゴリでは、各PGについて以下のメトリクスを用意した。

- STEP数 (ソースコードそのままの行数)
- 実STEP数 (STEP数からコメントと空行を除いたもの)
- LOC (ソースを展開したものの実STEP数)
- LOC.P (LOCの内、手続き記述部分)
- LOC.D (LOCの内、データ定義部分; LOC - LOC.P)
- セクション数 (セクション=プログラムより下位のロジックの集合; 他言語で言えば関数やメソッドに該当、以降Sec)

次に、モジュールが作成されてからの経過時間による減少の有無を調べるため、実際に対象アプリケーションのプログラムの年齢毎の障害発生の様子を図4に示す。縦軸は該当するプログラムにおいて40ヶ月の障害観測期間中に発生した障害の数を該当するプログラムの数で割ったものである。この図では、障害観測期間内に作成されたプログラム(月齢40未満)も計測対象に含めており、月齢40未満のプログラムはそのプログラムの障害発生数に(40/月齢)の重みを乗じる補正を施したものである。

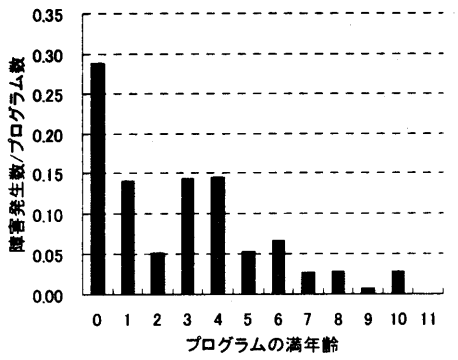

図4 プログラムの年齢と障害発生頻度

図より、プログラムが開発されてからの経過時間と障害発生率との関係が深いことが予想される。よって、時間カテゴリでは月齢を予測モデルのパラメータとして採用する。

プログラム内の複雑度としては、前述の通り、Mcを使用する。ただし、サイズ成分を取り除いたものが望ましい。このカテゴリでは、以下のメトリクスを用意した。

- McP (プログラム全体で測定したMc)
- McP / LOC.P
- McP / Sec
- 最大のMcS(セクション毎に測定したMc)
- McSが高いセクション数
- McSが高いセクションの割合 (以降、HMcR)

高いMcの基準としては文献[6]で紹介されているMc>10とした。

アーキテクチャ複雑度としては、インパクトスケール(以降、IS)を採用する。

各カテゴリからパラメータを1つずつ選んだ場合に最良(AIC最小)のモデルは、表の通りであった。

表1 フォールト率予測モデル

パラメータ	偏回帰係数	t値
(切片)	β_0 = -2.45	-7.73
月齢	β_1 = -0.0279	-8.10
Sec	β_2 = 0.0434	9.35
HMcR	β_3 = 3.18	4.14
IS	β_4 = 0.00114	5.43

各偏回帰係数の有意確率(P値)はすべて0.001以下であった。

各パラメータの寄与の比較は、t値(標準化偏回帰係数)の絶対値を比べればよい。寄与の大きさはSec>月齢>IS>HMcRの順であるが、IS、HMcRともサイズの

寄与の半分程度の寄与を持っていることが判った。

図 5にこの予測モデルが、対象のソフトウェア自身の障害率を予測した結果を示す。ここでは、6627 個の標本をその Z 値の昇順に 66 個のバケットに分けた。バケットの中の PG の障害の有無を平均して、そのバケットの障害発生率の実測値とした。バケットの障害発生率の予測値は、バケットの中の PG 毎のフォールト予測モデルの予測値の平均である。モデルが有効に働いていることが判る。

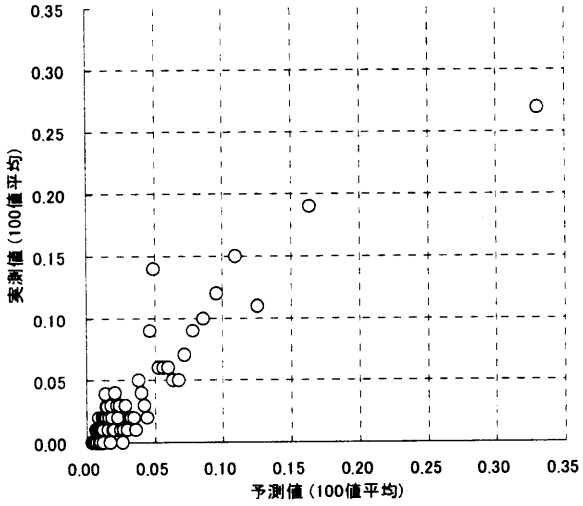

図 5 フォールト予測モデルの予測値と実測値

表 2は、標本データにおける各パラメータ間の相関係数である。これらは、独立であることが望ましいが、Sec と IS の間に弱い相関関係があることが判る。

表 2 パラメータ間の相関係数

	Sec	HMcR	IS
Sec	1		
HMcR	-0.0520	1	
IS	0.3158	-0.0226	1

そこで、Sec と IS の平面状で障害が実際にどのように分布するかを図 6に示した。Sec と IS を昇順の順位(1～6627)に変換し、238 個の障害を Sec 対 IS の 2 次元座標上にプロットしたものである。相関が強ければ、(0,0)点から右上がりの直線上に分布するはずであるが、全体的に(左上や右下の領域にまで)障害が分布していることが確認される。これは Sec が近い PG でも、IS が異なれば障害発生率は異なり、IS が PG の障害発生率に寄与していると予想できる。これは HMcR と Sec の関係でも同様であった。

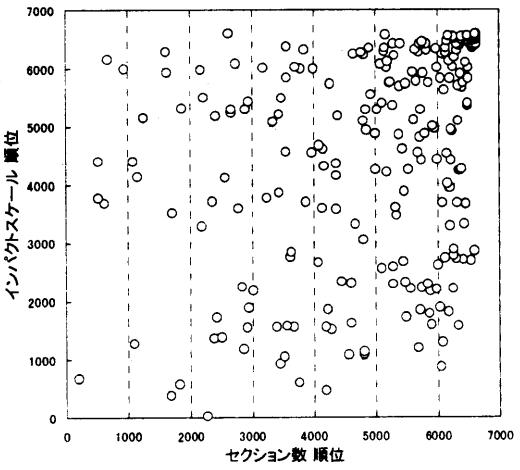

図 6 順位で見た障害分布

以上で、予測モデルが期待通りの結果を出していることが判った。

3.4 予測モデルの評価

次に、モデルの予測性能を評価する。Completeness (完全性)を、(障害を予測された障害 PG 数÷すべての障害 PG 数)と定義し、Correctness (正確性)を、(障害を予測された障害 PG 数÷障害を予測された全 PG 数)と定義する。図 7は予測モデルの、Completeness と Correctness である。

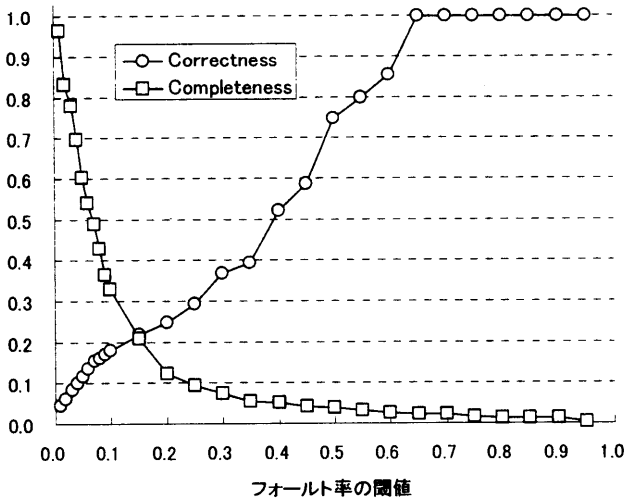

図 7 モデルの正確性と完全性

横軸は、障害の有無を判定するためのフォールト率の閾値である。モデルの出力値がこの値より大きければ障害有りとみなす。閾値を大きくすると Correctness は上昇するが、Completeness は低下してしまう。閾値が 0.5 の場合、Completeness は 0.038 なので実用価値が乏しいが、閾値を 0.1 とすると、Correctness は 0.182 まで落ちるが、Completeness は 0.328 なので、1/3 程度の障害を検出できることになる。

次に、このモデルの用途として、「障害が多そうなPGを優先してソースレビューを行う」という状況が考えられる。より少ないPG本数のレビューでより多い障害を発見できるほど意義が大きい。PD50という指標を、PGをフォールト予測率の降順に調べ、障害の50%を発見するまでに調べたPG数の百分率と定義する。

表3にパラメータを増減させた4種類のモデル毎のPD50の値を示す。これによれば、モデル(4)ではPG全体の12.0%をレビューすることで50%の障害を発見できる予測効果を持つ。一方、モデル(1)は16.5%のレビューが必要となる。これは、モデル(1)に高McCabeセクション率とインパクトスケールを追加することによって、37.5%の効率向上が果たせることを意味する。また、インパクトスケール単独による効率向上は24～26%であり、高McCabeセクション率よりもインパクトスケールの寄与の方が2倍程度大きい。

表3 レビュー効率

モデル	Sec	月齢	HMcR	IS	PD50
(1)	○	○	×	×	16.5
(2)	○	○	○	×	14.9
(3)	○	○	×	○	13.1
(4)	○	○	○	○	12.0

3.5 インパクトスケールの定義の検証

インパクトスケールは2.2節で仮に定義されたが、この定義の妥当性を確認する。まず、距離2以上が必要かについて調べる。予測モデルのパラメータのうち、インパクトスケールをS(M,d)に換えてそのAICを比較した結果を次表に示す。

アーキテクチャ複雑度	AIC	P値
IS	1756.915	<0.001
S(M,1)	1782.377	0.258
S(M,2)	1766.506	<0.001
S(M,3)	1755.986	<0.001
S(M,4)	1761.154	<0.001
S(M,5)	1777.617	0.014
S(M,6)	1781.653	0.639

この結果、距離1だけでは有意なモデルとはならず、距離2～4を含めなければ、効果の高いモデルとはならないことが判る。現在の距離10までの重ね合わせの定義は適当と言える。

次に、係数α_0=0.5の妥当性について調べる。IS(M,10,α)のαを変化させてモデルのAICを比較した結果、ちょうどα=0.5でAIC最小となった。よって、このソフトウェアに関する限り、α_0=0.5は妥当と言える。

4. まとめ

ソフトウェアのリスクを評価するためのメトリクス「インパクトスケール」を提案した。インパクトスケールは、ソースコードの保守の変更の影響波及範囲の大きさに基づいて、アーキテクチャの複雑度を表す尺度である。保守時の制限された情報から自動的に計測できるように、インパクトスケールはソースコードのみから計測できるように定義されている。その測定方法を述べ、実際に稼動している大規模COBOLアプリケーションにて実験と評価を行った。実験では、フォールト率予測モデルを作成し、モデルのパラメータにインパクトスケールを加えることで、フォールト率の予測能力が向上することを確認し、その有益性を示した。

4.1 今後の課題

本稿では、インパクトスケールとフォールト率との関連を示したが、インパクトスケールの増加がアプリケーションの劣化にどのように影響するかは示されていない。インパクトスケールの経年変化と生産性の変化の調査をする必要がある。

また、今回の実験では単一のアプリケーションのみで行われたため、生成したモデルの他アプリケーションへの適用する場合の問題点やカスタマイズの可能性などを調べる必要がある。

また、インパクトスケールにより問題点が指摘された場合、次の関心事はどのように改善を施すかである。インパクトスケールによって、アーキテクチャ的な問題点・改善点を指摘できないか、検討を行う。

文　献

[1] Arthur H. Watson, and Thomas J. McCabe, "Software complexity," Crosstalk, Journal of Defense Software Engineering, vol. 7, No. 12, pp.5—9, December 1994.

[2] 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).

[3] L. C. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++", Proc. of the 19th Int'l Conference on Software Eng., Boston, USA, pp.412—421 (1997).

[4] 神谷年洋, 楠本真二, 井上克郎, 「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」, オブジェクト指向'99 シンポジウム論文集, pp. 145—152.

[5] H. Akaike, "A new look at statistical model identification", IEEE Trans. on Automatic Control AU—19 716—722 (1974).

[6] W. S. Humphrey, "Managing the Software Process", Addison-Wesley (1989).