Osaka University Knowledge Archive

Title	ソフトウェアメトリックス値の変遷に基づいた注力すべきモジュールを特定する手法の提案
Author(s)	村尾, 憲治; 肥後, 芳樹; 井上, 克郎
Citation	電子情報通信学会論文誌D. 2008, J91-D(12), p. 2915-2925
Version Type	VoR
URL	https://hdl.handle.net/11094/26460
rights	copyright©2008 IEICE
Note	

Osaka University Knowledge Archive : OUKA

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

Osaka University

論　文

ソフトウェアメトリックス値の変遷に基づいた注力すべきモジュールを特定する手法の提案

村尾　憲治†　　肥後　芳樹†　　井上　克郎†

Identifying Problematic Modules Based on Metrics Transitions

Kenji MURAO†, Yoshiki HIGO†, and Katsuro INOUE†

あらまし　ソフトウェアの大規模化及び開発期間の短縮化により，ソフトウェアのすべてのモジュールに同様に力を注ぐことが難しくなってきている．特定のモジュールに力を注ぐ場合は，力を注ぐべきモジュールを特定しなければならないが，手動で行うと対象ソフトウェアの規模に応じて膨大な時間と労力を費やすことになる．そのため，開発者に負担をかけずに力を注ぐべきモジュールを特定することが求められる．本論文では，ソフトウェアメトリックス値の変遷を用いてソフトウェアの開発履歴を定量化することにより，力を注ぐべきモジュールを特定する手法の提案を行う．実際に提案手法を複数のオープンソースソフトウェアに対して適用した結果，将来問題の発生しやすいモジュールを，既存手法に比べて高い精度で特定することに成功した．

キーワード　バージョン管理システム，ソフトウェアメトリックス，バグ予測，品質評価

1. まえがき

ソフトウェアの大規模化，及び開発期間の短縮化により，ソフトウェアのすべてのモジュールに同様の力を注ぐことは現実的ではなくなってきている．例えば，問題が発生しやすいモジュールを知っていれば，そのモジュールに力を注ぐことで効率的な開発や保守が行え，またソフトウェアがどのような過程を経て開発されてきたのかを知っていれば，今後の開発の見通しも立てやすい．開発者がこのような知識を得るには，対象ソフトウェアについての調査が必要となるが，これを手動で行う場合は対象とするソフトウェアの規模に応じて膨大な時間と労力を費やすことになる．そこで，開発者にソフトウェアに関する知識を提供するための数多くの研究がなされている．

ソフトウェアの開発履歴をもとに知識を提供する研究が既に行われている．多くのものは CVS [1] や Subversion [2] などのバージョン管理システムを情報源としている．バージョン管理システムは，ソフトウェアの開発を効率良く管理するために用いられる．管理されるソフトウェアの開発履歴に関する情報は，バージョン管理システムのリポジトリというアーカイブに蓄積されている．このリポジトリに蓄積された開発履歴を閲覧することによって，ソフトウェアの開発過程についてより深い理解が得られることが知られている [3]．

一方，ソフトウェアのソースコードそのものをもとに知識を提供する研究も行われている．これに関しては，CK メトリックス [4] に代表されるソフトウェアメトリックスが有名である．特に，CK メトリックスは多くの研究において問題の発生しやすいモジュールを特定するのに有効であると示されている [5], [6]．

本論文では，メトリックス値の恒常性という，開発及び保守過程においてどの程度ソフトウェアメトリックスの値が安定しているかを表す概念を用いて，力を注ぐべきモジュールを特定する手法を提案する．提案手法は，各モジュールにおけるメトリックス値の恒常性を定量化しており，自動的に力を注ぐべきモジュールを特定することができる．また，提案手法を，将来バグ修正が多く行われるモジュールの特定に用いた結果についても記載する．

以降，**2.** では研究の動機について触れ，**3.** では本

† 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, 1–3 Machikaneyama–cho, Toyonaka–shi, 560–8351 Japan

論文における用語とメトリックス値の恒常性の計測手段について述べ，**4.** では提案手法の手順を説明する．**5.** では手法を実装したツールについて触れ，**6.** ではそのツールを用いて行った適用実験について述べる．更に，**7.** で関連する研究を挙げ，最後にまとめと今後の課題を **8.** に記す．

2. 動　　機

ソフトウェアのモジュールの性質や状態を表す指標としてソフトウェアメトリックスがある．これまでに，ソフトウェアの成果物を計測するための様々なメトリックスが提案されており，それらはいずれも最新の成果物（あるいは，過去のある時点における成果物）を計測する．つまり，既存のメトリックスは，ある時点における成果物の性質や状態を表しているが，それらがどのような過程を経て開発されたのかは表していない．筆者らは，成果物の現在の状態を表す情報と同様にそれらの進化の過程を表す情報が，今後の開発及び保守作業に有益であると考えている．

ソフトウェアのあるモジュールについて，複雑度メトリックスを計測した場合について考える．最新の成果物から複雑度メトリックスを計測した場合，最新の成果物がどの程度複雑であるかを把握することができる．しかし，そのモジュールが，開発当初より一定して複雑な状態にあるのか，開発が進むに従って複雑度が増しているのか，若しくは，変更が加わるたびに複雑度が増減しているのかは不明である．一般に複雑度が高いと，そのモジュールの理解や変更に大きな労力を要するといわれているが，一定して複雑な状態が維持されているのであれば，そのモジュールには大きな変更が加わっておらず，開発や保守のボトルネックになっているとは限らないといえよう．複雑度が単調に増加している場合は，改版のたびに新しい機能が加えられているのかもしれない．複雑度が増減している場合は，改版のたびに機能が削除・追加されたり，構造が変更されたりしていることを表しており，何か問題があるのかもしれない．

このように，モジュールのある時点における性質や状態だけでなく，その進化の過程を表現することにより，これまでは得ることができなかった新たな情報を得ることができるのではないかと考え，本提案手法を提案する．

3. 準　　備

ここでは，提案手法で用いる用語と，提案手法を読み進めるにあたり必要な知識を紹介する．

3.1 提案手法で用いる用語

3.1.1 メトリックス値の恒常性

メトリックス値の恒常性とは，開発及び保守過程において，ソフトウェアメトリックスの値がどの程度安定しているのかを表す概念である．「恒常性が高い」というのはメトリックス値が安定しており，「恒常性が低い」というのはメトリックス値が激しく変化していることを意味する．

3.1.2 スナップショット

スナップショットとは，ある時点においてソフトウェアを構成しているソースコードの集合である．ソフトウェアのソースコードに対して何らかの変更が加わると，新しいスナップショットが生成される．スナップショットは，CVS [1] や Subversion [2] などのバージョン管理システムのリポジトリに蓄積される情報，すなわち変更の日時や変更を行った開発者名，変更の前後における差分などを用いて生成することができる．

3.2 メトリックス値の恒常性を計測する方法

対象データのばらつきを表す指標として散布度，不確かさを表す指標としてエントロピー，変化を表す指標として距離などが挙げられる．これらは主として数学や統計学で用いられる指標である．提案手法では，これらの指標を用いてメトリックス値の恒常性を計測する．以降，これらの指標を用いた恒常性の計測手段について説明する．説明にあたり，以下の設定を用いる．

$MD = \{md_1, md_2, \cdots, md_\alpha\}$: 対象ソフトウェアにおけるモジュールの集合を表す．ただし，α は総モジュール数である．

$MT = \{mt_1, mt_2, \cdots, mt_\beta\}$: 用いるメトリックスの集合を表す．ただし，$\beta$ はメトリックスの総数である．

$CT = \{ct_1, ct_2, \cdots, ct_\gamma\}$: 対象ソフトウェアにおける変更時刻の集合を表す．ただし，γ は変更発生回数である．添字の値が小さいほど古い変更時刻を表す，つまり ct_1 は最初の変更時刻，ct_γ は最新の変更時刻を表す．

$v(i, j, k)$: 変更時刻 k におけるモジュール i のメトリックス j の値を表す．ただし，変更時刻 k においてモジュール i が存在しない場合はメトリックス値は，$v(i, j, k) = null$ とする．

3.2.1 エントロピー

Shannon の提唱した情報理論 [7] に依れば，エントロピーとは不確かさを表す指標である．メトリックス値の恒常性をメトリックス値の不確かさととらえることにより，エントロピーを恒常性の指標として用いることができる．

エントロピー H は対象モジュールの各メトリックスから導出される．モジュール i におけるメトリックス j のメトリックス値の集合，$\{v(i,j,1), v(i,j,2), \cdots, v(i,j,\gamma)\}$ において，値が *null* でないものの数を γ' とし，γ' 個のそれぞれの値を $v'_1, v'_2, \cdots, v'_{\gamma'}$ と表すことにする．更にその γ' 個の値のうち，異なる値の種類数を γ'' とする．γ'' 種類の値をそれぞれ $v''_1, v''_2, \cdots, v''_{\gamma''}$ とおくと，エントロピー $H(i,j)$ の以下の式で定義される．

$$H(i,j) = -\sum_{l=1}^{\gamma''} p_l \log_2 p_l \tag{1}$$

ここで p_l は値 v'_l の出現頻度を表し，次の式で定義される．

$$p_l = \frac{\sum_{k=1}^{\gamma'} \mathrm{equal}(v''_l, v'_k)}{\gamma''} \quad (1 \le l \le \gamma'')$$

ただし，

$$\mathrm{equal}(s,t) = \begin{cases} 1 & (s = t) \\ 0 & (s \ne t) \end{cases}$$

$\gamma'' = \gamma'$，つまりすべてのメトリックス値が異なる場合，エントロピーは $H(i,j) = -\log_2 \frac{1}{\gamma'}$ となり最大である．$\gamma'' = 1$，つまりすべてのメトリックス値が同じ場合，エントロピーは $H(i,j) = -\log_2 1 = 0$ となり最小である．

3.2.2 正規化エントロピー

エントロピー H の最大値は変更が行われた回数に依存するため，異なるソフトウェア間では正しく値を比較することが難しい．この問題を解決するため，新たに正規化エントロピーを定義する．

正規化エントロピー H' は，エントロピー H と同様に対象モジュールの各メトリックスから導出される．上記のエントロピー H で用いた設定に準じると，正規化エントロピー $H'(i,j)$ の以下の式で定義される．

$$H'(i,j) = \frac{H(i,j)}{-\log_2 \frac{1}{\gamma'}} \tag{2}$$

エントロピー $H(i,j)$ をその最大値で除算するため，正規化エントロピーは必ず $0 \le H'(i,j) \le 1$ となる．

3.2.3 四分位偏差

統計分野で用いられる散布度は，データのばらつきの程度を表現するものである [8]．メトリックス値の恒常性をメトリックス値のばらつきの程度ととらえることで，恒常性の指標として散布度を用いることができる．一般にメトリックス値が正規分布に従うとはいえないので，標準偏差を散布度として使うのは不適である．本論文では，代替として四分位偏差を用いる．

四分位偏差 Q は，対象モジュールの各メトリックスに対して値が得られる．モジュール i におけるメトリックス j のメトリックス値の集合，$\{v(i,j,1), v(i,j,2), \cdots, v(i,j,\gamma)\}$ における第 1 四分位数を q_1，第 3 四分位数を q_3 とすると，四分位偏差 $Q(i,j)$ は以下の式で定義される．

$$Q(i,j) = \frac{q_3 - q_1}{2} \tag{3}$$

3.2.4 四分位分散係数

四分位偏差 Q の値は絶対値であり，用いるメトリックスのスケール差に影響されてしまう．このため，異なるメトリックス間では正しく値を比較することが難しい．この問題への対処として四分位分散係数を用いる．

四分位分散係数 Q' は四分位偏差 Q と同様に対象モジュールの各メトリックスから導出される．モジュール i におけるメトリックス j のメトリックス値の集合，$\{v(i,j,1), v(i,j,2), \cdots, v(i,j,\gamma)\}$ における中位数を m とすると，四分位分散係数 $Q'(i,j)$ は以下の式で定義される．

$$Q'(i,j) = \frac{Q(i,j)}{m} \tag{4}$$

ただし $m = 0$ の場合は，

$$Q'(i,j) = \begin{cases} 0 & (Q(i,j) = 0) \\ \infty & (Q(i,j) \ne 0) \end{cases} \tag{5}$$

となる．

3.2.5 ハミング距離

情報理論 [7] においてハミング距離は，等しい文字数をもつ二つの文字列の中で対応する位置にある異なった文字の個数を意味する．文字列をメトリックス値の列に置き換えることで，ハミング距離を二つの変更間におけるメトリックス値の恒常性を示す指標として用いることができる．

ハミング距離 DH は対象モジュールの連続する二つの変更から導出される．ハミング距離 $DH(i,k)$ は以下の式で定義される．

$$DH(i,k) = \sum_{j=1}^{\beta} \mathrm{diff}(v(i,j,k-1), v(i,j,k)) \quad (6)$$

ただし，

$$\mathrm{diff}(s,t) = \begin{cases} 1 & (s \neq t) \\ 0 & (s = t)) \end{cases}$$

この式が示すように，ハミング距離 $DH(i,k)$ は変更時刻 ct_{k-1} と ct_k 間において値が変化したメトリックスの数を表している．したがってこの式が成り立つ条件は $\forall k((1 < k \leq \gamma) \wedge (v(i,j,k-1) \neq null) \wedge (v(i,j,k) \neq null))$ となる．この条件が成り立たない場合は，$DH(i,k) = null$ とする．

3.2.6 ユークリッド距離

ハミング距離 DH は変更前と変更後でメトリックス値が変化したかどうかのみを考慮しており，値がどの程度変化したかということは反映されない．それに対して，ユークリッド距離 DE はメトリックス値の変化の大きさを反映した恒常性を表す．メトリックス値の恒常性としてのユークリッド距離 DE は，用いるメトリックスの数，β 次元ユークリッド空間上の距離を表す [9]．ハミング距離 DH と同様に，ユークリッド距離 DE は対象モジュールの連続する二つの変更から導出される．

ユークリッド距離 $DE(i,k)$ は以下の式で定義される．

$$DE(i,k) = \sqrt{\vec{V}^T(i,k) \cdot \vec{V}(i,k)} \quad (7)$$

ただし，

$$\vec{V}(i,k) = \vec{v}(i,k) - \vec{v}(i,k-1),$$

$$\vec{v}(i,k) = [v(i,1,k), v(i,2,k), \cdots, v(i,\beta,k)]^T$$

ハミング距離 DH と同様に，この式が成り立つの条件は $\forall k((1 < k \leq \gamma) \wedge (v(i,j,k-1) \neq null) \wedge (v(i,j,k) \neq null))$ となる．この条件が成り立たない場合は，$DE(i,k) = null$ とする．

3.2.7 マハラノビス距離

ユークリッド距離を用いたメトリックス値の恒常性は，メトリックス間に相関がなく，かつスケール差もないとみなした状態での値となっている．しかし，実際にはメトリックス間に相関やスケール差の存在することが多い．例えば，異なる定義によりモジュールの結合度を求める二つのメトリックスがあった場合，目的を同じくするこの二つのメトリックスには強い正の相関関係があると考えられる．また，ソースコードの行数を表すメトリックスなどは変更によって絶対値が大きく変化することも珍しくないが，サブクラスの数を表すメトリックスなどは変更によって変化する絶対値があまり大きくなることは少ないと考えられる．そこで，相関やスケール差を考慮した値を求めることのできるマハラノビス距離も利用する．マハラノビス距離は多変量解析など統計学でよく用いられる距離の一種であり，情報分野においてもクラスタリング手法などにしばしば利用される [10], [11].

ハミング距離 DH やユークリッド距離 DE と同様に，マハラノビス距離 DM は対象モジュールの連続する二つの変更から導出される．また，マハラノビス距離 DM はその計測に対象のソフトウェア全体でのメトリックスの共分散行列 Σ を必要とする．この共分散行列は $\beta \times \beta$ の正方行列である．ユークリッド距離 DE における設定を用いて，要素に $null$ を含まないベクトル $\vec{v}(i,k)$ の個数を n $(1 \leq n \leq (\alpha \times \gamma))$ とする．この n 個のベクトルをそれぞれ $\vec{v}'_1, \vec{v}'_2, \cdots, \vec{v}'_n$ とおくと，メトリックスの共分散行列 Σ は以下の式で表すことができる．

$$\Sigma = \frac{1}{n} \sum_{l=1}^{n} (\vec{v}'_l - \vec{\mu}) \cdot (\vec{v}'_l - \vec{\mu})^T$$

ただし，

$$\vec{\mu} = \frac{1}{n} \sum_{l=1}^{n} \vec{v}'_l$$

この共分散行列 Σ を用いて，マハラノビス距離 $DM(i,k)$ は以下の式で定義される．

$$DM(i,k) = \sqrt{\vec{V}(i,k)^T \Sigma^{-1} \vec{V}(i,k)} \quad (8)$$

この式が示すように，マハラノビス距離はメトリックスの共分散行列 Σ の逆行列を計算に用いる．したがって，共分散行列 Σ が特異行列（非正則行列）となる場合はマハラノビス距離の計測自体を行わないものとする．また，ハミング距離 DH やユークリッド距離 DE と同様に，この式が成り立つの条件は $\forall k((1 < k \leq \gamma) \wedge (v(i,j,k-1) \neq null) \wedge (v(i,j,k) \neq null))$ となる．この条件が成り立たない場合は，$DM(i,k) = null$ とする．

4. メトリックス値の恒常性を算出する手順

以下の手順により，メトリックス値の恒常性を計測する．なお，恒常性を計測するためには，対象ソフトウェアがバージョン管理システムを利用して開発されていなければならない．

計測手順 1：過去のスナップショットの取得
計測手順 2：メトリックス値の計測
計測手順 3：メトリックス値の恒常性の算出
以下では，各手順について詳しく説明する．

計測手順 1：過去のスナップショットの取得

バージョン管理システムに保存された分析対象のソフトウェアの変更履歴情報を利用し，過去すべての変更時刻におけるスナップショットを取得する．

図 1 は，ソフトウェアの変更履歴の例を示したものである．この図では，ソフトウェアを構成する四つのソースファイル F_1，F_2，F_3，F_4 が縦軸に配置されている．横軸は時刻を表し，時刻 ct_1，ct_2，ct_3，ct_4 において変更が行われている．また，$F_{1,ver.1}$ や $F_{4,ver.3}$ などはソースファイルとそのバージョンを示している．図 1 に示すようなソフトウェアの変更履歴があったとき，各変更時刻におけるスナップショットは表 1 のようになる．

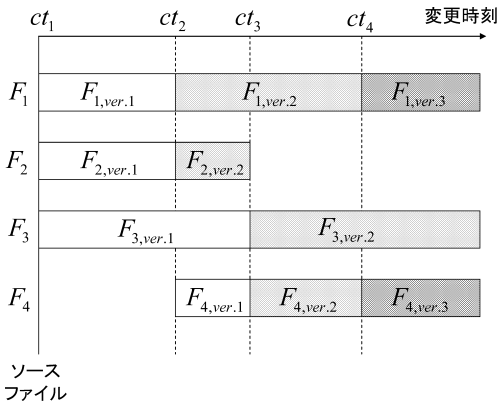


図 1　ソフトウェアの変更履歴の例
Fig. 1　Example of a software development history.

表 1　図 1 に対応するスナップショット
Table 1　Snapshots for Fig. 1.

変更時刻	スナップショット
ct_1	$\{F_{1,ver.1}, F_{2,ver.1}, F_{3,ver.1}\}$
ct_2	$\{F_{1,ver.2}, F_{2,ver.2}, F_{3,ver.1}, F_{4,ver.1}\}$
ct_3	$\{F_{1,ver.2}, F_{3,ver.2}, F_{4,ver.2}\}$
ct_4	$\{F_{1,ver.3}, F_{3,ver.2}, F_{4,ver.3}\}$

計測手順 2：メトリックス値の計測

手順 1 で取得したすべてのスナップショットを対象にメトリックス値の計測を行う．

用いるメトリックスはモジュールの単位や目的に合わせて選ぶ必要がある．例えば，モジュールの単位をクラスとしたいのであればクラスを対象としたメトリックスを，ファイルとしたいのであればファイルを対象としたメトリックスを用いる．あるいは，結合度に注目したいのであれば結合度を表すメトリックスを，凝集度に注目したいのであれば凝集度を表すメトリックスを用いる．後述の適用事例では，モジュールの単位をクラスとし，主に CK メトリックス [4] を用いている．

計測手順 3：メトリックス値の恒常性の算出

手順 2 で計測したメトリックスに対して，**3. 2** で紹介した各計測手段を適用することにより，各モジュールにおけるメトリックス値の恒常性を求める．**3. 2** で紹介した計測手段は，モジュールとメトリックス，若しくはモジュールと変更時刻の二つの次元をもっている．提案手法では，これらの値をモジュールではない次元に対して和をとったものを，そのモジュールのメトリックス値の恒常性として定義する．モジュール md_i の恒常性 $\omega(i)$ は，以下の式で表される．

$$\omega(i) = \begin{cases} \sum_{j=1}^{\beta} \omega(i,j) & (\omega = H, H', Q, Q') \\ \sum_{k=1}^{\gamma} \omega(i,k) & (\omega = DH, DE, DM) \end{cases} \quad (9)$$

ただし，ω は H，H'，Q，Q'，DH，DE，DM のいずれかである．また，四分位偏差はメトリックス値の絶対値をもとに算出されるため，用いるメトリックス間にスケール差が存在する場合は，上記の式を用いてモジュールの恒常性を求めることは適切ではない．

5. 実　　装

4. で述べた計測手順を自動で行うツールを実装した．実装したツールの特徴は以下のとおりである．

- 対象は Java 言語で記述されたソフトウェア．
- 対応しているバージョン管理システムは CVS．
- CK メトリックスをもとにして，恒常性を計測する．CK メトリックスは対象とするモジュールがクラスであるので，実装したツールでもクラスをモジュールの単位として扱う．
- 実装したツールは，CVS のリポジトリを入力とし，モジュールの恒常性を CSV 形式のファイルと

表 2　対象ソフトウェアの概要
Table 2　Overview of the target software systems.

ソフトウェア名	FreeMind	JHotDraw	HelpSetMaker
種別	ダイアグラム生成ツール	ドローツール	ドキュメント生成ツール
開発言語	Java	Java	Java
開発者数	12	24	2
総スナップショット数 γ	104	196	260
最初の変更時刻 ct_1	2000/08/01 19:56:09	2000/10/12 14:57:10	2003/10/20 13:05:47
最後の変更時刻 ct_γ	2004/02/06 06:04:25	2005/04/25 22:35:57	2006/01/07 15:08:41
ct_1 におけるソースファイル数	67	144	14
ct_γ におけるソースファイル数	80	484	36
ct_1 におけるソースコードの総行数	3,882	12,781	797
ct_γ におけるソースコードの総行数	14,076	60,430	9,167

して出力する．

6. 適用実験

ここでは，**5.** で述べたツールを用いて行った実験について述べる．

6.1 実験目的

実験の目的は，提案手法を用いることにより，力を注ぐべきモジュールを特定できるかを調査することである．**2.** で述べたように，メトリックス値の恒常性が低いクラスでは，そうでないクラスに比べて問題が発生しやすいと考えられる．この実験では，力を注ぐべきモジュールを「将来多数のバグが発生するモジュール」として，そのようなモジュールを特定することができるかを調査した．

この実験は，特に以下の点において調査することにより，提案手法の汎用性を示すことに重点をおく．

（1） 単一のソフトウェアだけでなく，複数のソフトウェアに対しても力を注ぐべきモジュールを特定することができるか．

（2） 開発の特定の期間だけでなく，様々な期間に適用しても力を注ぐべきモジュールを特定することができるか．

以降，項目 1 を調査するための実験を「実験 1」と呼び，項目 2 を調査するための実験を「実験 2」と呼ぶ．

6.2 実験対象

実験対象は，FreeMind [12]，JHotDraw [13]，HelpSetMaker [14] の三つのオープンソースソフトウェアである．これらのソフトウェアの概要を表 2 に示す．

6.3 利用したメトリックス

この実験で用いたソフトウェアメトリックスは，CK メトリックス（RFC，CBO，LCOM，NOC，DIT）とクラスの行数（LOC）の計六つである．特に CK メトリックスはバグの予測に有効であるとの報告がなされており [5], [6]，CK メトリックスによる予測とメトリックス値の恒常性による予測を比較することで，提案手法の有効性が示される．

6.4 実験方法

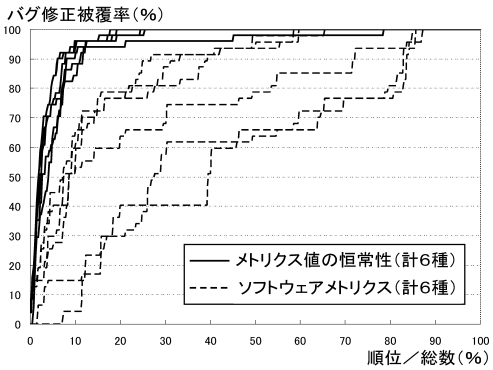
実験は，次に示す手順で行った．なお，これ以降，バグを修正するための変更をバグ修正変更，バグ修正変更の対象となったファイルの延べ数をバグ修正数と呼ぶ．また，この実験では，内部クラスは測定の対象外である．

実験手順 1：対象ソフトウェアのスナップショット群を，学習用データと調査用データに分割する．分割は変更回数に基づいて行われ，古いスナップショット群が学習用データに，新しいスナップショット群が調査用データに分類される．

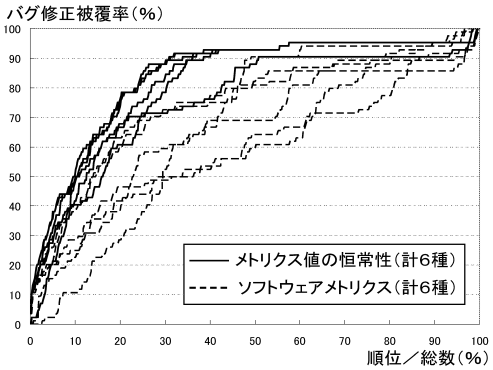
実験手順 2：学習用データのスナップショット群から，メトリックスを計測し，恒常性を算出する．

実験手順 3：調査用データのスナップショット群から，バグ修正変更を特定する．この実験では，バグ修正変更の特定は，CVS リポジトリに記録されているログメッセージをもとに行った．例えば FreeMind の場合，バグ修正変更の際のメッセージログには，文頭に「Bug fix:」と記載する習慣がある．この実験では，ログメッセージに「bug」と「fix」の両文字列（大文字小文字は区別しない）が含まれているかどうかでバグ修正変更かどうかを判断した．

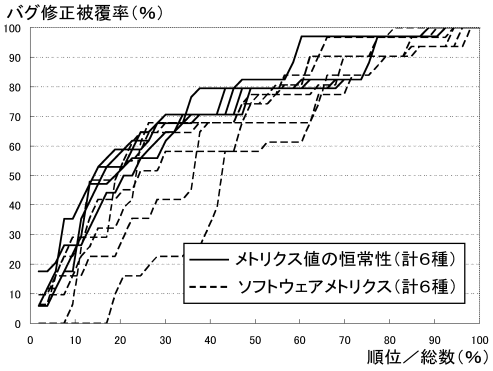
実験手順 4：学習用データから得たメトリックス値の恒常性と調査用データから得たバグ修正変更を突き合わせる．具体的には，対象クラスをメトリックス値の恒常性の降順に並べ，上位 x%のクラスにおけるバグ修正の被覆率（y%）を調査する．同様に，対象クラスを学習用データの最後のスナップショットにおけるメトリックス値の降順に並べ，上位 x%のクラスにおけ



(a) FreeMind



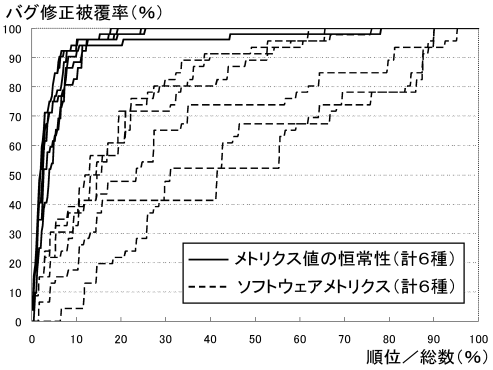
(b) JHotDraw



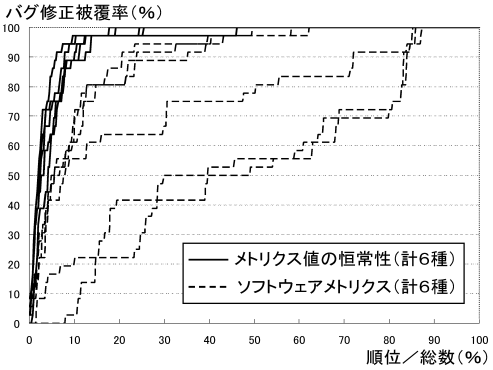
(c) HelpSetMaker

図 2　複数ソフトウェアにおける学習メトリックスと恒常性のバグ予測比較結果（学習用データ 1/3・調査用データ 2/3 での分割）

Fig. 2　Comparison result between metrics constancy and raw metrics for multiple software systems. (The leaning data is anterior one third, and the investigating data is posterior two third)



(a) 学習用データ 1/4・調査用データ 3/4 での分割



(b) 学習用データ 1/2・調査用データ 1/2 での分割

図 3　異なる分割点における学習メトリックスと恒常性のバグ予測比較結果（対象は FreeMind）

Fig. 3　Comparison result between metrics constancy and raw metrics for different application timings for FreeMind.

るバグ修正の被覆率（y%）も調査する．

6.5 結　　果

実験 1 では，学習用データは前 1/3 のスナップショット群とした．実装したツールを用いて，**4.** で述べた計測手順 1〜3 を実行するのに要した時間は，FreeMind が約 28 分，JHotDraw は約 40 分，HelpSetMaker は約 18 分であった(注1)．

図 2 は，各対象ソフトウェアにおける実験手順 4 の結果を示している．この図では，恒常性のグラフが実線で表され，メトリックスのグラフが点線で表されている．例えば，FreeMind の場合（図 2 (a)）では，どの恒常性の指標を用いても，上位 20%のクラスにおいて，調査用データのバグ修正変更のうちのおよそ 97〜100%が行われていることが分かる．一方，メトリックスの場合は，上位 20%のクラスにおいて，およそ 22〜89% のバグ修正変更が行われている．この図より，す

(注1)：実験は，CPU：1.86 GHz，メモリ：2 GByte を搭載した Windows マシン上で行った．なお，対象ソフトウェアの CVS リポジトリは，あらかじめ実験マシンのローカルストレージに複製した上でツールを実行した．

べての対象ソフトウェアにおいて，ソフトウェアメトリックスそのものを用いるよりもメトリックス値の恒常性を用いた方が，力を注ぐべきモジュールを精度良く特定できていることが分かる．

実験 2 では，前 1/2，1/3，1/4 をそれぞれ学習用データとして用いた．図 3 は，FreeMind に対する実験手順 4 の結果を表している(注2)．図 3 と図 2 (a) より，適用時期にあまり依存することなく，力を注ぐべきモジュールを精度良く特定できていることが分かる．一方，ソフトウェアメトリックスは，適用時期が早いほど（古いスナップショットのメトリックス値を用いるほど），精度が落ちているのが分かる．

実験 1 と実験 2 より，以下に示す結論を導くことができる．

- すべての恒常性がバグ予測に有効であり，指標間の差異もそれほど大きくなかった．
- ソフトウェアメトリックスは，メトリックス間で予測の差が大きかった．つまりメトリックスにもバグ修正に有効なものがあった．

この実験より，メトリックス値が激しく変化しているクラス（メトリックス値の恒常性が低いクラス）は，そうでないクラスに比べて将来多くのバグが発生しているといえる．また，メトリックス値の恒常性を用いることにより，そのようなクラスを自動的に特定できることを示した．

7. 関 連 研 究

本研究の関連としてソフトウェアメトリックスを用いた研究とバージョン管理システムを利用した研究をいくつか紹介する．

7.1 ソフトウェアメトリックスを用いた研究

メトリックス値の恒常性と同様に，ソフトウェアメトリックスそのも力を注ぐべきモジュールの特定に用いられている．

本論文の実験において用いた CK メトリックスは，Chidamber ら [4] によって生み出されたオブジェクト指向プログラムを対象としたメトリックスである．Basili らは，八つの学生チームによるオブジェクト指向プログラミング言語 C++を用いて開発された情報管理システムを対象とし，CK メトリックスと他のメトリックスとの比較実験を行った [5]．その実験結果から，他のメトリックスに比べ，CK メトリックスはより多くのソフトウェアの欠陥を，よりソフトウェア開発の早期において予測できることを示した．また，Subramanyam らは，産業界で開発された八つのソフトウェアに対して CK メトリックスの有効性を調査した [6]．その結果，Basili らと同様，CK メトリックスとソフトウェアの欠陥に大きな関係があることを見出した．同時に，ソフトウェアによってメトリックスの有効性が異なることも示した．

Nagappan らはソフトウェアのバグを予測する最も良いメトリックスがソフトウェアによって異なることを発見し，対象ソフトウェアにおける過去のバグに関する情報からソフトウェアメトリックスによってバグを予測する回帰モデルを構築した [15]．彼らはいくつかの大規模な商用ソフトウェアに手法を適用し，その有効性を評価している．

本提案手法とこれらの手法との違いは，情報源が異なることである．本提案手法ではバージョン管理システムを情報源としているが，上記の手法の情報源はソースコードである．ただし，Nagappan らの手法はソースコードに加えてバグに関する情報が管理されている必要がある．そのため，正確にバグに関する情報を管理しているソフトウェアが少ないのが問題となっている．適用可能なソフトウェアが多いという点では，ソースコードのみを必要とする CK メトリックスが最も優れているといえる．しかし，近年バージョン管理システムを用いているソフトウェア開発は非常に多いため，本提案手法が適用できるソフトウェアも多い．また，本論文は，本提案手法は CK メトリックスよりも有効なバグ修正の予測が可能であることを実験で確かめた．

7.2 バージョン管理システムを用いた研究

近年では，バージョン管理システムなどの開発履歴に関する情報を利用してソフトウェアの特性を把握する試みも増えている．

Williams らは Apache ウェブサーバにおける過去のバグに関する情報を調べ，メソッドの戻り値をチェックすることがバグの発生を予測するのに有効であると考えた [16]．そこで，メソッドの戻り値チェックに基づいた将来のバグの発生を予測する手法について，最新のソースコードだけを用いる手法と，それに加えてバージョン管理システムから得られる履歴情報も利用する手法の比較を行った．その結果，最新のソースコードだけでなくバージョン管理システムの情報を併

(注2)：前 1/3 を学習データとして用いた場合のグラフは図 2 で示しているので，図 3 からは省いている．

用した手法の方がより優れた予測を導き出すことができ，バージョン管理システムに含まれる情報を有用性を示した．

Hassan らは情報理論 [7] によるエントロピーをソフトウェアの開発過程に当てはめ，ソフトウェアの開発履歴に含まれる情報量の計測を試みた [17]．彼らはこの情報によりソフトウェアの開発過程の監視が行えるとし，特にプロジェクトマネージャにとって有益な手法であることを述べた．

Śliwerski らはバージョン管理システムと過去のバグに関する情報とを関連づけ，バグを発生させる原因となった変更の特定を行っている [18]．このような変更が頻繁に行われるモジュールは，今後もバグを発生させやすいと考えられる．開発者このモジュールを提示することで，将来の開発や保守を効率的に行うことが望めるとしている．

Ostrand らはバージョン管理システムと過去のバグに関する情報から負の二項回帰モデルを構築し，対象ソフトウェアの次のリリース時にバグを発生するファイルの予測を試みた [19]．彼らはいくつかの適用事例から負の二項回帰モデルに基づく予測結果が単にコードの行数に基づく予測よりも有効であることを示し，この手法によってテスト時に注意すべきソースファイルの提示が可能であるとした．

花川はソフトウェアメトリックスを用いたモジュール間の結合度と論理的なモジュール間の結合，すなわち同時に変更が施されるモジュール群に着目し，新たな複雑度の指標を定義した [20]．また，ソフトウェアの開発過程におけるこの複雑度の変化を可視化し，ソフトウェアの複雑さの変化を視覚的に把握するツールを作成している．

Vasa らはソフトウェアの開発過程におけるリリース間でのモジュールごとの利用数，被利用数の変化を調査している [21]．彼らは複数のオープンソースソフトウェアに対して実験を行い，利用数，被利用数の変化から得られるソフトウェアの特性の取得を試みた．その結果，ほとんどのモジュールは開発が経過するにつれ利用数，被利用数の変化が見られなくなってくるが，利用数，被利用数が非常に多い一部のモジュールは開発が経過しても利用数，被利用数が変化し続ける，などといったソフトウェアの特性を分析した．

Willams らの研究は，本提案手法との立場が近いが，彼らの手法は関数の戻り値に関するバグに特化しているという点で異なる．Hassan らによる手法は本提案手法と同じくバージョン管理システムを情報源としているが，本提案手法ではモジュールに関する情報が得られるのに比べ，彼らの手法で得られる情報は対象のソフトウェア全体についてのものである点が異なる．また，Śliwerski らによる手法と Ostrand らによる手法はバグに関する情報が管理されているソフトウェアを対象とするため，手法を適用できるソフトウェアが少ないのが問題となっている．花川による手法と Vasa らによる手法は，本提案手法と同じ情報源を用い，得られるソフトウェアの特性の対象も本提案手法に近い．しかし，花川による手法ではソフトウェアの特性の対象となるメトリックスは結合度のみであり，Vasa らによる手法でも対象となるメトリックスは利用数，被利用数という一種の結合度のみである．本提案手法では，適用事例においては CK メトリックスを中心としたメトリックスを用いたが，手法としてはソースコードを対象とする定量的なメトリックスであればどんなメトリックスであっても用いることができる．また，本提案手法ではメトリックス値の恒常性を数値として出力する点も既存の研究には見られない特徴である．

8. む す び

本論文では，バージョン管理システムのリポジトリに蓄積された開発管理情報から，力を注ぐべきモジュールを特定する手法を提案した．提案手法は，ソフトウェアメトリックスを用いているが，メトリックス値そのものではなく，メトリックスの値がどのように変化してきたかに着目することにより，力を注ぐべきモジュールの特定を行う．

更に提案手法を実装し，複数のオープンソースソフトウェアに対して，将来バグを多数発生させるモジュールの特定を行う実験を行った．その結果，CK メトリックスに比べ，より高い精度でそのようなモジュールの特定を行えることが確認された．

提案手法を応用することにより，バグ修正の実効的な予測が行えると期待される．例えば，以下のような応用が考えられる．

- プロジェクトの途中でモジュールの変動度やメトリックス値の時系列値の計測結果からバグを多数含むモジュールを検出する．
- プロジェクト自体の失敗の可能性を分析し，ソフトウェア開発のリスク管理を行う．

今後の課題として重要なのは，手法を実装したツールの拡張である．現状では制限が多く，以下に挙げる

ような機能が望まれる．

- C++や C# など，他のプログラミング言語への対応
- Subversion など，他のバージョン管理システムへの対応
- 他の様々なソフトウェアメトリックスへの対応
- 他のバグ修正の予測手法との比較

謝辞 本研究は一部，文部科学省「次世代 IT 基盤構築のための研究開発」（研究開発領域名：ソフトウェア構築状況の可視化技術の開発普及）の委託に基づいて行われた．

文　献

[1] “CVS,” http://ximbiot.com/cvs/wiki/
[2] “Subversion,” http://subversion.tigris.org/
[3] P.H. Feiler, “Configuration management models in commercial environments,” Technical Report CMU/SEI-91-TR-7 ESD-9-TR-7, Software Engineering Institute, Carnegie Mellon University, 1991.
[4] S. Chidamber and C. Kemerer, “A metric suite for object-oriented design,” IEEE Trans. Softw. Eng., vol.25, no.5, pp.476–493, June 1994.
[5] V.R. Basili, L.C. Briand, and W.L. Melo, “A validation of object-oriented design metrics as quality indicators,” IEEE Trans. Softw. Eng., vol.22, no.10, pp.751–761, Oct. 1996.
[6] R. Subramanyam and M.S. Krishnan, “Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects,” IEEE Trans. Softw. Eng., vol.29, no.4, pp.297–310, April 2003.
[7] C.E. Shannon, “The mathematical theory of communication,” Bell Syst. Tech. J., vol.27, pp.379–423, 623–656, July 1948.
[8] 宮川公男，基本統計学，有斐閣，1999.
[9] 中村幸四郎，寺阪英孝，伊東俊太郎，池田美恵，ユークリッド原論，共立出版，1996.
[10] F. Fessant, P. Aknin, L. Oukhellou, and S. Midenet, “Comparison of supervised self-organizing maps using euclidian or mahalanobis distance in classification context,” Proc. 6th International Work-Conference on Artificial and Natural Neural Networks, pp.637–644, June 2001.
[11] C.P. Tenorio, F. de A. T. de Carvalho, and J.T. Pimentel, “A partitioning fuzzy clustering algorithm for symbolic interval data based on adaptive mahalanobis distances,” Proc. 7th International Conference on Hybrid Intelligent Systems, pp.174–179, Sept. 2007.
[12] “FreeMind,” http://freemind.sourceforge.net/wiki/
[13] “JHotDraw,” http://www.jhotdraw.org/
[14] “HelpSetMaker,” http://www.cantamen.com/helpsetmaker.php
[15] N. Nagappan, T. Ball, and A. Zeller, “Mining metrics to predict component failures,” Proc. 28th International Conference on Software Engineering, pp.452–461, May 2006.
[16] C.C. Williams and J.K. Hollingsworth, “Automatic mining of source code repositories to improve bug finding techniques,” IEEE Trans. Softw. Eng., vol.31, no.6, pp.466–480, June 2005.
[17] A.E. Hassan and R.C. Holt, “Studying the chaos of code development,” Proc. 10th Working Conference on Reverse Engineering, pp.123–133, Nov. 2003.
[18] J. Śliweski, T. Zimmermann, and A. Zeller, “HATARI: Raising risk awareness,” Proc. 10th European Software Engineering Conference, pp.107–110, Sept. 2005.
[19] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, “Predicting the location and number of faults in large software systems,” IEEE Trans. Softw. Eng., vol.31, no.4, pp.340–355, April 2005.
[20] 花川典子，“論理結合とモジュール結合による複雑度を用いたソフトウェア進化の可視化，” 第 14 回ソフトウェア工学の基礎ワークショップ，pp.65–74, Nov. 2007.
[21] R. Vesa, J.G. Schneider, and O. Nierstrasz, “The inevitable stability of software change,” Proc. 23rd IEEE International Conference on Software Maintenance, pp.4–13, Oct. 2007.

（平成 20 年 4 月 2 日受付，7 月 6 日再受付）

村尾　憲治

平 18 阪大・基礎工・情報卒．平 20 同大大学院博士前期課程了．現在，大阪府総務部に勤務．在学時，ソフトウェア保守支援及びリポジトリマイニングに関する研究に従事．

肥後　芳樹　（正員）

平 14 阪大・基礎工・情報中退．平 18 同大大学院博士後期課程了．日本学術振興会特別研究員を経て，平 19 阪大・情報・コンピュータサイエンス・助教．博士（情報科学）．コードクローン分析・リファクタリングに関する研究に従事．情報処理学会，IEEE 各会員．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校・情報工学科・助教授．平元阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．博士（工学）．平 14 阪大・情報・コンピュータサイエンス・教授．ソフトウェア工学の研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE，ACM 各会員．