RB-004

# メトリクス値の変遷を用いた Fault-Prone モジュール特定手法の改良

# Improving Identification Method of Fault-Prone Modules Based on Metrics Transitions

**西野 稔** †　**肥後 芳樹** †　**楠本 真二** †
Minoru Nishino　Yoshiki Higo　Shinji Kusumoto

## 1　まえがき

ソフトウェアの開発，保守において，将来問題の発生しそうなモジュールを特定すれば，開発や保守の効率を高めることが出来ると期待される．ところが，特に大規模なソフトウェアにおいて，手動でそのようなモジュールを特定するには，膨大な時間と労力を要することになる．そこで，開発者に問題の発生しやすいモジュールを知らせるため，数多くの研究が行われている．

そのうちの 1 つとして，開発履歴におけるソフトウェアメトリクスの値の変遷を解析して，力を注ぐべきモジュールを特定する手法が提案されている [1]．現在のところ，小規模のソフトウェアのみに対してその有効性が示されており，実規模のソフトウェアに対しては評価されていない．そこで本研究では，この手法をより規模の大きいソフトウェアに対して適用して有効性を示すとともに，この手法を改良し，利用する履歴情報の選別や変更時期による重み付け，および実装方法の改良を行って，有効性を評価している．

## 2　準備

ここでは，本論文で用いる用語と，本論文を読み進めるにあたり必要な知識を紹介する．

### 2.1　スナップショット

スナップショットとは，ある時点でソフトウェアを構成している全てのソースコードの集合である．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 4 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 $F_{1,rev.1}$ や $F_{4,rev.3}$ などはソースファイルとそのリビ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．

表 1　図 1 に対応するスナップショット

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

† 大阪大学大学院情報科学研究科

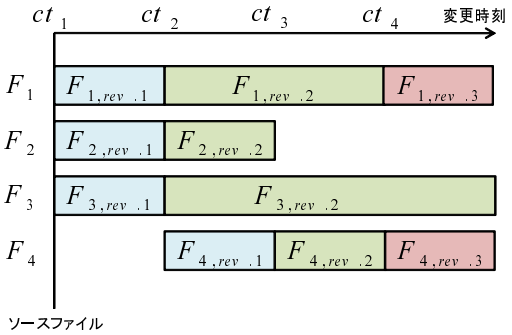



図 1　ソフトウェアの変更履歴の例

過去のスナップショットは，バージョン管理システムのリポジトリに蓄積される各ファイルのリビジョンの情報，すなわち変更の日時とその時点でのソースコードを用いて生成することができる．あるいは Subversion[2] のように，バージョン管理システムの機能により日時もしくはリビジョン番号の指定のみで，スナップショットが生成できるものもある．

### 2.2　ソフトウェアメトリクス

ソフトウェアメトリクスとは，ソフトウェアの様々な特性 (複雑度，信頼性，効率など) を表現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 [3]．例えば，ソフトウェア開発の設計や実装の段階において，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，それによって後のエラーの発生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 [4]．

本研究において言及するのは，ソースコードを対象とする複雑度メトリクスのみとする．以降，ソフトウェアメトリクスを単にメトリクスを記載する．

### 2.3　メトリクス値の恒常性

メトリクス値の恒常性とは，複数のスナップショット間でメトリ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」とはメトリクス値が安定しており，「恒常性が低い」とはメトリクス値が激しく変化していることを意味する．

一般にメトリクス値が高いほど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るが，メトリクス値だけでなく恒常性も高い場合は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る．一方，恒常性が低いモジュールは，なんらかの問題を含んでいる可能性を

考えることができる．

### 2.4 メトリクス値の恒常性を算出する統計手法

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．文献 [1] の手法では，これらの指標を用いてメトリクス値の恒常性を計算する．以降，これらの指標を用いた恒常性の計算方法について説明する．説明にあたり，以下の設定を用いる．

$MD = \{md_1, md_2, \ldots, md_\alpha\}$ 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，$\alpha$ は総モジュール数である．

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

$CT = \{ct_1, ct_2, \ldots, ct_\gamma\}$ 対象ソフトウェアの変更時刻の集合を表す．ただし，$\gamma$ は変更回数であり，添字が小さいほど古い変更時刻を表す．なお，変更時刻とは，対象ソフトウェアにおいて 1 つ以上のソースコードが追加，削除，変更された時刻を指す．

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

なお算出手法として，エントロピー，正規化エントロピー，四分位偏差，四分位分散係数，ハミング距離，ユークリッド距離，マハラノビス距離を用いている．このうちエントロピー，正規化エントロピー，四分位偏差，四分位分散係数は，対象モジュールの各メトリクスに対して導出され，以降の説明においてはモジュール $md_i$ のメトリクス $mt_j$ から算出したそれぞれの値を，$H(i,j)$，$H'(i,j)$，$Q(i,j)$，$Q'(i,j)$ と記述する．ハミング距離，ユークリッド距離，マハラノビス距離は，対象モジュールの連続する 2 つの変更から導出され，モジュール $md_i$ の変更時刻 $ct_{k-1}$ および $ct_k$ から算出したそれぞれの値を，$DH(i,k)$，$DE(i,k)$，$DM(i,k)$ と記述する．具体的な計算方法は既存手法を説明した論文 [1] において説明されているので，ここではエントロピーとハミング距離についてのみ説明する．

#### 2.4.1 エントロピー

Shannon の提唱した情報理論 [5] に依れば，エントロピーとは不確かさを表す指標である．エントロピーが高いほどメトリクス値の変動が激しく，恒常性が低いとみなすことが出来る．

エントロピーは対象モジュールの各メトリクスに対して導出される．モジュール $md_i$ におけるメトリクス $mt_j$ の値の集合 $\{v(i,j,1), v(i,j,2), \ldots, v(i,j,\gamma)\}$ において，値が $null$ でないものの個数を $\gamma'$ とし，$\gamma'$ 個のそれぞれの値を $v'_1, v'_2, \ldots, v'_{\gamma'}$ と表すことにする．更にその $\gamma'$ 個の値のうち，異なる値の種類数を $\gamma''$ とし，$\gamma''$ 種類の値をそれぞれ $v''_1, v''_2, \ldots, v''_{\gamma''}$ とおくと，エントロピー $H(i,j)$ は以下の式で定義される．ただし式中の対数の底としては，何を選んでも本質的な差異はない．5 章における実験のための実装では，底として 2 を用いた．

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

ただし，

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

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

$\gamma'' = \gamma'$，つまり全てのメトリ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．$\gamma'' = 1$，つまり全てのメトリクス値が同じ場合，エントロピーは $H(i,j) = -\log_2 1 = 0$ となり最小である．

なお，$\{v(i,j,1), v(i,j,2), \ldots, v(i,j,\gamma)\}$ が全て $null$ 値である場合，エントロピー $H$ は定義されないので，$H(i,j) = null$ とする．

#### 2.4.2 ハミング距離

情報理論 [5] においてハミング距離は，等しい文字数をもつ 2 つの文字列の間で対応する位置にある異なった文字の個数を意味する．文字列をメトリクス値の列に置き換えることで，ハミング距離を 2 つの変更間におけるメトリクス値の変動の激しさを示す指標として用い，ハミング距離が大きい場合は恒常性が低く，ハミング距離が小さい場合は恒常性が高いとみなすことができる．

ハミング距離は対象モジュールの連続する 2 つの変更から導出される．モジュール $md_i$ の変更時刻 $ct_{k-1}, ct_k$ 間のハミング距離 $DH(i,k)$ は以下の式で定義される．

$$DH(i,k) = \sum_{j=1}^{\beta} \mathrm{diff}(v(i,j,k-1), v(i,j,k)) \qquad (2)$$

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 $ct_k$ の間に値が変化したメトリクスの数を表している．したがって，$v(i,j,k-1) = null$ または $v(i,j,k) = null$ となる変更時刻 $ct_k$ については計算できないため，この場合 $DH(i,k) = null$ とする．

## 3 既存手法

文献 [1] において提案されている手法では，以下の手順によりメトリクス値の恒常性を算出する．

**手順 1** 過去のスナップショットの取得
**手順 2** メトリクス値の計測
**手順 3** メトリクス値の恒常性の算出

以下では，各手順について説明する．

### 3.1 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蓄積された対象ソフトウェアの情報から，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．

### 3.2 手順 2: メトリクス値の計測

手順 1 で取得したすべてのスナップショットを対象に，メトリクス値の計測を行う．用いるメトリクスはモ

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するのであればクラスを対象としたメトリクスを，ファイルとするのであればファイルを対象としたメトリクスを用いる．あるいは，結合度に注目するのであれば結合度を表すメトリクスを，凝集度に注目するのであれば凝集度を表すメトリクスを用いる．

### 3.3 手順 3: メトリクス値の恒常性の算出

手順 2 で計測したメトリクス値は，モジュール，メトリクス，変更時刻の 3 つの次元を持っている．これらに対して，2.4 節で紹介した各算出手段を適用することにより，モジュールとメトリクス，もしくはモジュールと変更時刻の 2 つの次元を持つ値が算出される．その後，これらの値をモジュールではない次元に対して和をとることで，モジュールの恒常性を示す値を算出することができる．すなわち，モジュールの恒常性を示す値 $\omega(i)$ は，以下の式で表わされる．

$$\omega(i) = \begin{cases} \sum_{j=1}^{\beta} \omega(i,j) & (\omega = H, H', Q, Q') \\ \sum_{k=1}^{\gamma} \omega(i,k) & (\omega = DH, DE, DM) \end{cases} \quad (3)$$

ただし，$\omega$ は $H, H', Q, Q', DH, DE, DM$ のいずれかである．また，2.4 で紹介した算出手段はいずれも，メトリクス値の変動が大きく，不安定であるほど大きい値を取るため，上記の式で求められた $\omega(i)$ について，$\omega(i)$ が大きいならば「恒常性が低い」，$\omega(i)$ が小さいならば「恒常性が高い」という意味になる．

## 4 改良点

本章では，3 章で説明した従来の手法に対する改良点を述べる．本論文で提案する改良点は大きく分けて，計測方法の改良と，実装の改良に分けられる．

### 4.1 計測方法の改良

計測方法に対する改良は，以下の 2 点である．

- タグ単位のメトリクス値の変遷に基づいた計測
- 変更日時による重み付け

#### 4.1.1 タグ単位のメトリクス値の変遷に基づいた計測

従来の手法では，ソフトウェアの全ての変更時刻におけるメトリクス値から恒常性を算出している．そのため変更回数が数千にもなるような大規模ソフトウェアに適用した場合，多大な計算時間を要する．それに対して，変更時刻のうち一部のみを利用すれば，正確性は劣るものの計算時間は大幅に短縮できると期待される．そこで，いくつかの変更時刻を選び出してこの手法を適用することを考えた．

利用する変更時刻は，開発者によってタグを設定された変更を用いることとした．タグが設定されるのは，一般的にリリース時など特別な変更をチェックインする際であり，これらの変更は全体の中でも重要度が高いと考えられるためである．

#### 4.1.2 変更日時による重み付け

従来の手法では，全ての変更におけるメトリクス値からエントロピー等の指標を計算し，モジュールの恒常性を算出している．これに対し，変更の時期によって重み付けを行い，恒常性を算出できるように改良した．これにより，図 2 のようにメトリクス値が変遷するモジュール $md_1$ と $md_2$ があった場合に，より有効な予測ができると考えられる．

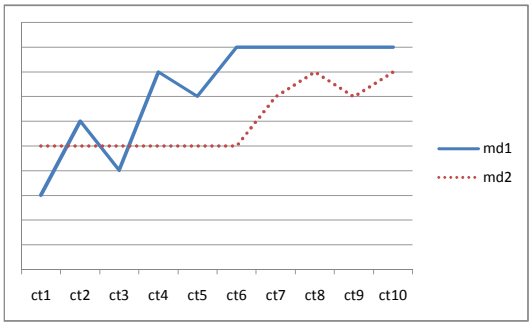



図 2　重み付けが有効なメトリクス値の変遷

図 2 において，$md_1$ は開発の初期段階でメトリクス値が大きく変動しているが，最近の変更では安定している．それに対して $md_2$ のメトリクス値は，変動の大きさは $md_1$ より小さいものの，最近の変更で頻繁に変動している．この場合，従来手法では $md_1$ の方が恒常性が低く問題を含む可能性が高いと判断する．しかし最近の変更でメトリクス値が安定しているため，既に問題が取り除かれたと考えることもできる．そのため，$md_1$ に力を注ぐよりは最近の変更でメトリクス値が頻繁に変化している $md_2$ に注目する方がより効果的であるといえる．そのため，変更の時期による重み付けにより，$md_2$ のようなモジュールを発見できるようにした．

### 4.2 実装の改良

実装に対する改良は，以下の 2 点である．

- バージョン管理システム Subversion[2] への対応
- マルチスレッドプログラミングによる処理の高速化

#### 4.2.1 バージョン管理システム Subversion への対応

従来の手法では，手法そのものは利用するバージョン管理システムに依存せずに適用できるが，実装において対象とするバージョン管理システムを CVS[6] に限定していた．それに対して，現在では多くのプロジェクトがバージョン管理に Subversion を用いている．そこで，新たに Subversion に対応することにより，実用性が高まると考えられる．

#### 4.2.2 マルチスレッドプログラミングによる処理の高速化

本手法の課題の 1 つに，処理時間がチェックアウト回数に比例するため，大規模ソフトウェアに対して適用する場合に膨大な時間を要する点が挙げられる．そこで，マルチスレッドプログラミングを用いて一部の処理を並列化することにより，処理時間の削減を行った．

## 5 適用実験

本手法を実装したツールを用いて行った実験について述べる．

### 5.1 実験目的

実験の目的は，提案手法よる力を注ぐべきモジュールの特定が実規模ソフトウェアに対して有効であるかの調査と，4 章で説明した改良による効果の測定である．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，メトリクス値の恒常性を用いることにより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．

### 5.2 実験内容

#### 5.2.1 実験項目

この実験では，以下の 3 項目を調査した．

**実験 1** 恒常性を用いた場合とメトリクス値をそのまま用いた場合との，バグ予測効果の比較

**実験 2** タグ単位の情報から算出した恒常性を用いた場合とメトリクス値を用いた場合との比較

**実験 3** 重み付けによる改善効果の確認

**実験 4** マルチスレッドプログラミングによる高速化の効果の確認

#### 5.2.2 実験対象ソフトウェア

実験対象として，Apache Ant[7] というオープンソースソフトウェアを用いた．このソフトウェアの概要を表 2 に示す．

#### 5.2.3 モジュールの単位と利用したメトリクス

適用実験においては，モジュールの単位をクラスとした．ただし内部クラスは測定の対象外としている．用いたメトリクスは，クラスの行数 (LOC) および CK メトリクス (RFC，CBO，LCOM，NOC，DIT)[8] である[*1]．実験 1 および実験 2 における比較対象としても，これらのメトリクスを用いる．これらのメトリクスを用いる理由は，一般的に用いられるメトリクスであり，特に CK メトリクスについてはその有効性が報告されているためである [9].

#### 5.2.4 実験手順

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に行った変更をバグ修正変更，あるクラスがバグ修正変更の対象となった回数をバグ修正回数と呼ぶ．

**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．本実験では，前 1/3 のスナップショットを学習用データとした．

**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クス値を計測し，恒常性を算出する．

**実験手順 3** 調査用データの期間に含まれる，バグ修正変更を特定する．

**実験手順 4** 学習用データから得たメトリ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．具体的な評価方法は 5.2.5 節で説明する．

表 2 実験対象ソフトウェアの概要

| ソフトウェア名 | Apache Ant |
|---|---|
| 種別 | ビルドツール |
| 開発言語 | Java |
| 開発者数 | 34 |
| 総スナップショット数 $\gamma$ | 5,491 |
| 最初の変更時刻 $ct_1$ | 2000/1/13 19:42:41 |
| 最後の変更時刻 $ct_\gamma$ | 2009/1/19 21:27:38 |
| $ct_1$ におけるファイル数 | 355 |
| $ct_\gamma$ におけるファイル数 | 1,182 |
| $ct_1$ における総行数 | 2,785 |
| $ct_\gamma$ における総行数 | 187,323 |

[*1] CK メトリクスの WMC を用いていないが，これは実験に用いたメトリクス計測ツールにおいて未実装であったためである．

#### 5.2.5 評価方法

実験結果の評価には，以下の二種類の方法を用いた．

**バグ修正の被覆率** 各指標により全てのクラスを昇順に並べ，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．

**順位の比較** バグ数，および各指標により全てのクラスを順位付けし，各指標による順位とバグ数による順位 (正解の順位) の差を比較する．

### 5.3 各実験の適用結果と考察

#### 5.3.1 実験 1: メトリクス値をそのまま用いる場合との比較

この実験では，メトリクス値の恒常性を用いた場合と，メトリクス値をそのまま用いた場合で，結果がどのように変化するのかを比較した．

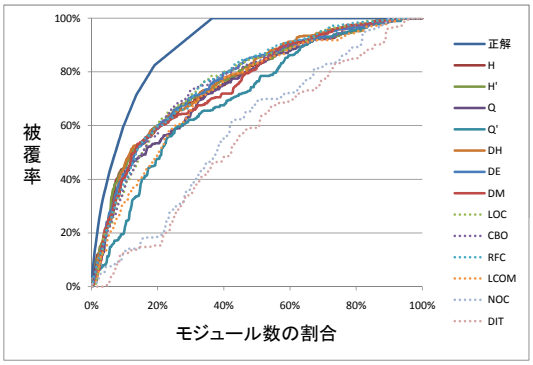
図 3(a) は被覆率のグラフであり，順位付けに用いる指標を変えて，上位 $x\%$ のクラスにおけるバグ修正の被覆率 $y\%$ をプロットしている．「正解」とは対象クラスを実際のバグ修正回数の降順に順位付けした結果である．H，H’，Q，Q’，DH，DE，DM は各指標を用いた恒常性により順位付けした結果を表し，LOC，CBO，RFC，LCOM，NOC，DIT は各メトリクス値をそのまま用いた場合の結果を表す (以降の図においても同様である)．

図 3(a) では $x$ 軸の範囲を 0% から 100% としているが，実験目的は問題の多いモジュールを発見できるかの調査であるため，重要なのは上位に予測したクラスに含まれるバグの割合である．そこで，$x$ 軸の上位 20% を拡大したグラフが図 3(b) である．

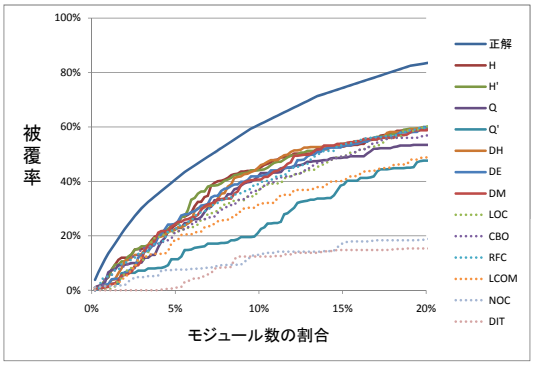
図 3(b) より被覆率において，メトリクス値の恒常性を用いる場合とメトリクス値をそのまま用いる場合の最大値を比較すると，いずれも上位 20% のクラスに約 60% のバグが含まれているなど，大差がないことがわかる．しかし，各指標を個別に調べると，メトリクス値の恒常性を用いた場合どの指標もある程度バグ予測に有効であるのに対し，メトリクス値には有効とはいえないものがあった．

また，それぞれの指標による予測について，被覆率では大差がない場合でも，予測される順位は異なる．そのため，各指標の長所を上手く組み合わせることで，より有効な予測ができるのではないかと考えられる．この仮説を補強するため，図 4 に示す調査を行った．

図 4 は，予測された順位と正解の順位との差について，正解との差が $x$ 以内のクラス数 $y$ をプロットしたグラフである．「併用」とは，恒常性とメトリクス値全ての指標の中で，正解との差の最小値をとり，その値が $x$ 以内になるクラス数を表す．なお，クラス数については，バグ数が 1 以下のクラスは数えていない．これは指標によっては，最下位となる順位がバグ数 1 や 0 の順




(a) 全体




(b) 上位 20%

図 3　実験 1: 恒常性とメトリクス値による被覆率

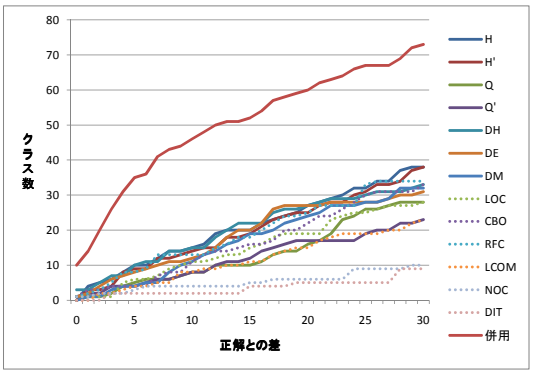



図 4　実験 1: 正解との差が $x$ 以内のクラス数

位と近い値になり，良い結果が得られているように見えるためである．具体例として，バグ数 1 の順位 (97 位) と，NOC が 0 の順位 (108 位) が挙げられる．最下位が何位になるか，すなわち最下位となるクラスの個数は，計算結果のスケールや離散的か連続的かなどの計算方法の性質に依存するため，バグの少なさを予測したわけではない．

図 4 より読み取れる事として，各指標を個別に用いて誤差が 10 位以内に収まったクラスの数は最大でも 15 クラスだが，いずれかの指標で誤差が 10 位以内に収まったクラスの数は 40 クラスを超える．また，個別に用いた場合で 5 位以内のクラスは最大で 10 クラスであるのに対し，いずれかの指標で 5 位以内となったクラスは 30 クラス以上ある．これらの点から，ある指標で的確な予測ができないクラスに対しても，別の指標を用いることでより有効な予測ができることがわかる．

以上の結果より本手法は，メトリクス値を用いた調査と組み合わせることで，より適切に問題の起こりやすいモジュールを判別することができるといえる．

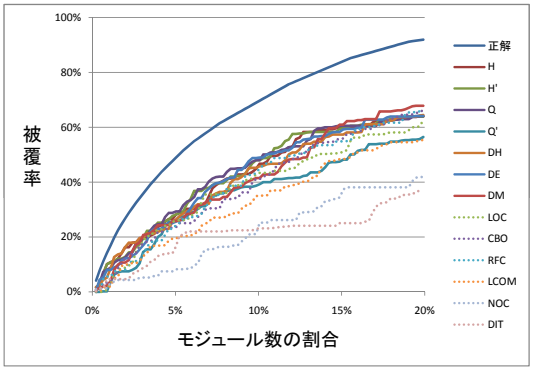



図 5　実験 2: タグ単位の情報による被覆率 (上位 20%)

### 5.3.2　実験 2: タグ単位の適用による比較

この実験では，4.1.1 節で説明したタグ単位でのメトリクス値の変遷に基づいた恒常性による予測の精度を調査した．実験 1 と同様に，タグ単位での 1/3 の分割点において，メトリクス値の恒常性を用いた場合と，メトリクス値をそのまま用いた場合の結果を比較した．

図 5 は被覆率の上位 20% 部分を拡大したグラフである．この図より，恒常性とメトリクス値の最大値を見ると，上位 20% のクラスに 65% 程度のバグが含まれており，実験 1 と同様にこれらは同程度バグ予測に有効であると結論付けることができる[*2]．各指標を個別に見ると，やはりメトリクス値にはバグ予測に対する有効性が比較的低いものがあった．

また，本実験においても，それぞれの指標により予測される順位に差がある．そこで，予測された順位と正解の順位との差について，正解との差が $x$ 以内のクラス数 $y$ を求め，図 6 に示した．ただし，バグ数 4 の順位 (36 位) と NOC の最下位 (57 位) が近いため，実験 1 と同様の理由でバグ数が 4 以下のクラスは数えていない．

図 6 より実験 1 と同様に，ある指標を単独で用いて正しく予測できないクラスを，別の指標により効果的に予測することができるとわかる．

なお，予測に用いた変更数は全ての変更を用いた実験 1 が 1829 個で，この実験が 16 個である．また，計算時間は実験 1 が 22 時間，この実験が 40 分であり，大幅に短縮できたことがわかる[*3]．

以上より，タグ単位のメトリクス値の変遷を用いた手

[*2] メトリクス値をそのまま用いた場合の結果が実験 1 と異なるのは，全リビジョンの 1/3 の分割点とタグ単位の 1/3 の分割点が一致しないためである．

[*3] この計算時間は 4.2.2 節の改良を施す以前の測定結果である．

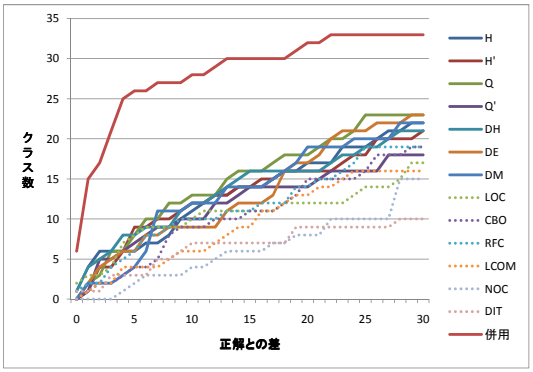



図 6　実験 2: 正解との差が $x$ 以内のクラス数

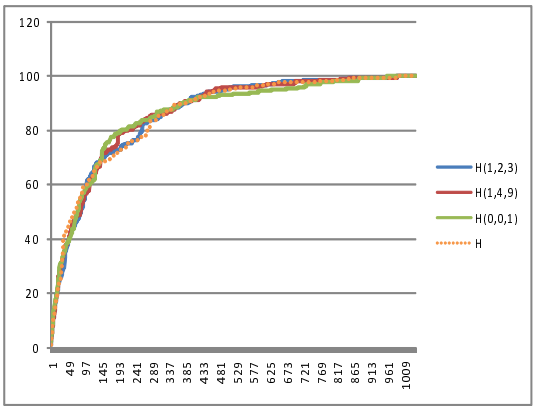



図 7　実験 3: エントロピーによる被覆率

法は，全ての変更時刻を用いる場合と同等の結果を期待でき，かつ計算時間を短縮できるといえる．

#### 5.3.3　実験 3: 重み付けの有無による比較

この実験では，4.1.2 節で説明した変更日時による重み付けが，有効であるかどうかを確認する．

重み付けの範囲を広く取るため，前 2/3 のスナップショットを学習用データとした．重み付けは学習用データを変更回数でさらに 3 等分し，それぞれ 1:2:3，1:4:9，および 0:0:1 の重みをつけて実験した．

結果についてはエントロピーについてのみを示すが，他の指標についても同様の結果が確認できた．図 7 はエントロピーによる被覆率のグラフである．図中の H はエントロピーを重み付けなしで計算した場合の結果を表し，H(a,b,c) は a:b:c の重み付けで適用した場合の結果を表す．

グラフより，重み付けによりバグ予測の精度はほとんど変化していないことがわかる．そのため本手法において，今回実験した限りにおいては，重み付けによる結果の改善は確認できなかったといえる．

#### 5.3.4　実験 4: 高速化の効果の確認

この実験では，4.2.2 節で説明したマルチスレッドプログラミングによる高速化の効果を測定する．

5.3.2 節でも述べたが，改良前の実験 1 の計算時間が 22 時間である．それに対して，改良後に実験 1 と同様の実験を行ったところ，16 時間で終了した．この結果から，改良により計算時間が短縮されたことが確認できる．

## 6　まとめ

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履歴の情報から，力を注ぐべきモジュールを特定する手法について，既存手法の提案時には不十分であった規模の大きいソフトウェアに対する実験を行った．その結果，本手法は対象ソフトウェアの規模に関わらず，有益な情報を得られることが確認された．また，既存手法について複数の改良を行い，その効果を測定した．

今後の課題として，第一に恒常性とメトリクス値を併用した，力を注ぐべきモジュールの特定が挙げられる．それに加えて，モジュールの粒度の変更や対応するプログラミング言語の追加も考えている．

## 謝辞

本研究は，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記述の開発普及) の委託に基づいて行われている．

また，文部科学省科学研究費補助金基盤研究 (C)(課題番号:20500033) の助成を得て行われている．

## 参考文献

[1] 村尾憲治, 肥後芳樹, 井上克郎．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．電子情報通信学会論文誌, Vol. J91-D, No. 12, pp. 2915–2925, 2008.

[2] Version control with subversion. available online at <http://svnbook.red-bean.com/>.

[3] Mark Lorenz and Jeff Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice Hall, 1994.

[4] Maurizio Pighin and Roberto Zamolo. A predictive metric based on discriminant statistical analysis. In *Proceedings of the 19th International Conference on Software Engineering*, pp. 262–270, 1997.

[5] Claud E. Shannon. The mathematical theory of communication. *Bell System Technical Journal*, Vol. 27, pp. 379–423,623–656, 1948.

[6] Karl Fogel and Moshe Bar. *Open Source Development with CVS*. The Coriolis Group, 2000.

[7] Apache ant project. available online at <http://ant.apache.org/>.

[8] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 476–493, 1994.

[9] Victor R. Basili, Lionel C. Briand, and Walcelio L. Melo. A validation of object-oriented designmetrics as quality indicators. *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, pp. 751–761, 1996.