RB-005

変数使用に着目した Fault-Prone メソッド特定手法の提案と評価

Proposal and Evaluation of Fault-Prone Methods Identification Technique Based on Variable Usages

兼光 智子 †　肥後 芳樹 †　楠本 真二 †
Tomoko KANEMITSU　Yoshiki HIGO　Shinji KUSUMOTO

1 まえがき

近年，ソフトウェア開発の大規模化・複雑化・開発期間の短縮化に伴い，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困難になってきている．Fault-Prone モジュール，つまりフォールトを含んでいる可能性が高いモジュールを予測し，そのモジュールに力を注ぐことで効率よく開発や保守が行える．

一般的な Fault-Prone モジュールの予測手法では，ソフトウェア保守性を評価するための複雑度メトリクスの値を説明変数として予測モデルを作成し，予測したいモジュールに適用する．しかし，予測に使用するメトリクス値を測定するモジュールの単位は，ファイルやクラスの場合が多い．モジュールの単位は小さい方が，問題を特定しやすくより適切な箇所に力を注ぐことができ開発や保守の効率が上がる．

そこで，本研究では変数使用情報に着目し，Fault-Prone メソッドを特定する手法を提案する．具体的には，変数の使用可能な範囲と実際に使用されている範囲に着目し，その差が大きいものほどフォールトが含まれる可能性が高いと考える．

次に，提案手法を実際のソースコードに適用しフォールト情報との関連を見ることで提案手法の有効性を調査した．その結果，変数使用情報を用いた予測とフォールトとの相関は高く，従来のメトリクスであるコード行数やサイクロマチック数よりフォールトとの相関があることを確認した．

2 準備

2.1 複雑度メトリクス

この研究では，ソースコードの複雑さを計測するメトリクスを単に複雑度メトリクスとよぶ．複雑度メトリクスは多く提案されており，代表的な複雑度メトリクスとして以下のものがある．

2.1.1 コード行数

基本的な複雑度メトリクスとして行数がある．ソースコードの行数が多いということはそれだけ規模が大きく，ゆえに複雑であるといえる．空行とコメントのみの行を除いた行をもってコード行数とすることが多い．

2.1.2 サイクロマチック数

サイクロマチック数とは，McCabe によって提案されたメトリクスである [1]．プログラム制御の流れを有向グラフで表現したときの枝の数 e，節点の数 n を用いて $e-n+2$ で表される．この値は直観的にはソースコードの分岐の数に 1 を加えた数を表す．サイクロマチック数が多いと，テストケース（ホワイトボックステスト）を作成する手間がかかり，保守性が下がると指摘されている．サイクロマチック数は経験的に 10 以下に抑えることが望ましいといわれている [2]．

2.1.3 CK メトリクス

CK メトリクスとは，Chidamber と Kemerer らの提案したメトリクスである [3]．オブジェクト指向ソフトウェアに対する代表的なメトリクスであり，クラスの複雑度を静的に評価する．計測する内容ごとに WMC, LCOM, DIT, NOC, CBO, RFC という 6 つのメトリクスが定義されている．全てのメトリクスにおいて，計測値が大きいほうが，複雑なクラスであり好ましくないことを示す．

2.2 Fault-Prone

Fault-Prone とは，フォールトを含んでいる確率が高いという意味である．ソフトウェア開発において，Fault-Prone モジュールを予測する研究が広く行われている．例えば Basili らは，CK メトリクス [3] を用いて，ソフトウェア開発の早期段階で Fault-Prone クラスの予測が行えることを示した [4]．

Fault-Prone モジュールを特定できれば，テスト工程において，Fault-Prone モジュールにより多くのテスト工程を割り当てることで，無駄なテスト工程を省き，ソフトウェア開発の効率化を図れ信頼性も向上する [5]．Arisholm らは，レガシーシステムにおいて過去のバージョンのフォールトや変更情報を用いて新しいバージョンにおける Fault-Prone モジュールを予測したとき，大きくテスト効率が上がったと報告している [6]．

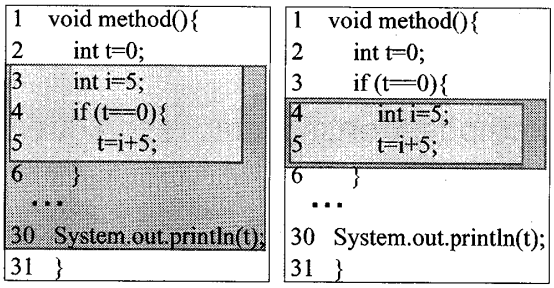
モジュールから計測されたメトリクス値（コード行数，サイクロマチック数等）を説明変数とし，モジュールのフォールトの有無を目的変数とする Fault-Prone モジュール予測モデルが多数提案されている．予測に使用するメトリクス値を測定するモジュールの単位は，ファイルやクラスの場合が多い．

2.3 ソフトウェアの保守性

ソフトウェアの品質には様々なものがあるが，ISO/IEC9126 ではソフトウェアの保守性を次のように定義している [7]．

保守性
　修正のしやすさに関する能力．修正は，是正もしくは向上，又は環境の変化，要求仕様の変更及び機能仕様の変更にソフトウェアを適応させることを含めてもよい．

† 大阪大学大学院情報科学研究科



(a) 差が大きい場合　　(b) 差が小さい場合

図 1: 変数のスコープと実際に使用されている範囲の例
濃い網掛は変数のスコープを表し，薄い網掛は実際に使用されている範囲を表す

一般に，ソフトウェアの保守性を評価するためには複雑度メトリクスやクローンメトリクスが用いられることが多い [8][9].

3 提案手法

変数の使用可能な範囲と実際に使用されている範囲について考える.

例として図 1(a) のメソッドを考える．変数 i は 3 行目で宣言されており，30 行目まで使用可能である．しかし，実際に最後に使用されているのは 5 行目である．このように変数の使用可能な範囲と実際に使用されている範囲には差がある場合がある.

変数 i は if 文の中でのみ使用されているので，図 1(b) のように変数 i の宣言を if 文の中で行うとする．変数 i は 4 行目で宣言されており，5 行目まで使用可能である．実際に最後に使用されているのは 5 行目である．このように変数の宣言位置を変更することによって，変数の使用可能な範囲と実際に使用されている範囲の差が小さくなる.

変数の使用可能な範囲と実際に使用されている範囲について以下のように定義する.

変数のスコープ
変数の使用可能な範囲であり，図 1(a) の変数 i では，3 行目から 30 行目である.

変数が実際に使用されている範囲
変数が宣言されてから最後に使用されるまでであり，図 1(a) の変数 i では，3 行目から 5 行目である.

変数が実際に使用されている範囲は常に変数の使用可能な範囲に含まれる．変数の使用可能な範囲に比べ変数が実際に使用されている範囲が極端に小さい場合，変数のスコープが無駄に広く変数の誤った使用が起こる可能性がある.

変数の使用可能な範囲と実際に使用されている範囲の差を評価する指標として次の 2 つを定義する.

変数使用行率
変数のスコープのうち実際に使用されている範囲の割合であり，式 (1) のように定義する．割合をとることで，実際に使用されている範囲の局所性を表すことができる.

$$\text{変数使用行率} = \frac{\text{実際に使用されている範囲の行数}}{\text{変数のスコープ行数}} \quad (1)$$

この値が小さいほど，変数の使用可能な範囲と実際に使用されている範囲の差が大きくなる.
図 1(a) の変数 i では，実際に使用されている範囲の行数は 3 行，変数のスコープ行数は 28 行なので，変数使用行率は

$$\text{変数使用行率} = \frac{3}{28} \simeq 0.107$$

よって，0.107 となる.

変数使用行差
変数の使用可能な範囲と実際に使用されている範囲との差の行数であり，式 (2) のように定義する．絶対的な行数をとることで，変数が使用されていないことにより影響を受ける実際の行数を表すことができる.

$$\text{変数使用行差} = \text{変数スコープ行数} - \text{実際に使用されている範囲の行数} \quad (2)$$

この値が大きいほど，変数の使用可能な範囲と実際に使用されている範囲の差が大きくなる.
図 1(a) の変数 i では，変数のスコープ行数は 28 行，実際に使用されている範囲の行数は 3 行なので，変数使用行差は

$$\text{変数使用行差} = 28 - 3 = 25$$

よって，25 となる.

変数使用行率が小さい変数や変数使用行差が大きい変数を含むメソッドの方がフォールトが含まれる可能性が高いと考える.

変数使用行率・変数使用行差は変数ごとに算出する値であるが，1 つのメソッドには複数の変数が含まれる．メソッドの特性としての値を 1 つに決めるため，その変数ごとの値から平均をとるなどして算出する．変数のスコープ行数が小さいと変数使用行率は大きく変数使用行差は小さくなる．しかし，変数のスコープ行数が大きいほど与える影響は大きいと考え，通常の平均だけでなく変数のスコープ行数で重みを付けた加重平均と変数使用行率では最低値，変数使用行差では最大値も評価する．各値を次のように定義する．なお各値の例として図 1(a) のメソッド内の変数を使用する．変数 i の変数使用行率は 0.107，変数使用行差は 25，スコープ行数は 28 であり，変数 t の変数使用行率は 1.0，変数使用行差は 0，スコープ行数は 29 である.

平均変数使用行率
変数ごとの使用行率の和を変数の数で割る

$$\text{平均変数使用行率} = \frac{1}{n}\sum_{i=1}^{n} \text{変数 } a_i \text{の変数使用行率} \quad (3)$$

図 1(a) のメソッドでは，

$$\text{平均変数使用行率} = \frac{1}{2}(0.107 + 1.0) = 0.5535$$

よって，0.5535 となる．

加重平均変数使用行率

変数の使用行率にその変数のスコープ行数をかけたものの和を変数のスコープ行数の和で割る

$$\text{加重平均変数使用行率} = \frac{\sum_{i=1}^{n}(\text{変数 } a_i\text{の変数使用行率} \times \text{変数 } a_i\text{のスコープ行数})}{\sum_{i=1}^{n} \text{変数 } a_i\text{のスコープ行数}} \quad (4)$$

図 1(a) のメソッドでは，

$$\text{加重平均変数使用行率} = \frac{0.107 \times 28 + 1.0 \times 29}{28 + 29} \simeq 0.5613$$

よって，0.5613 となる．

最低変数使用行率

メソッドに含まれる変数の使用行率の最低値

図 1(a) のメソッドでは，

$$min(0.107, 1.0) = 0.107$$

よって，0.107 となる．

平均変数使用行差

変数ごとの使用行差の和を変数の数で割る

$$\text{平均変数使用行差} = \frac{1}{n}\sum_{i=1}^{n} \text{変数 } a_i\text{の変数使用行差} \quad (5)$$

図 1(a) のメソッドでは，

$$\text{平均変数使用行差} = \frac{1}{2}(25 + 0) = 12.5$$

よって，12.5 となる．

加重平均変数使用行差

変数の使用行差にその変数のスコープ行数をかけたものの和を変数のスコープ行数の和で割る

$$\text{加重平均変数使用行差} = \frac{\sum_{i=1}^{n}(\text{変数 } a_i\text{の変数使用行差} \times \text{変数 } a_i\text{のスコープ行数})}{\sum_{i=1}^{n} \text{変数 } a_i\text{のスコープ行数}} \quad (6)$$

図 1(a) のメソッドでは，

$$\text{加重平均変数使用行差} = \frac{25 \times 28 + 0 \times 29}{28 + 29} \simeq 12.287$$

よって，12.287 となる．

最大変数使用行差

メソッドに含まれる変数の変数使用行差の最大値

図 1(a) のメソッドでは，

$$max(28, 0) = 28$$

よって，28 となる．

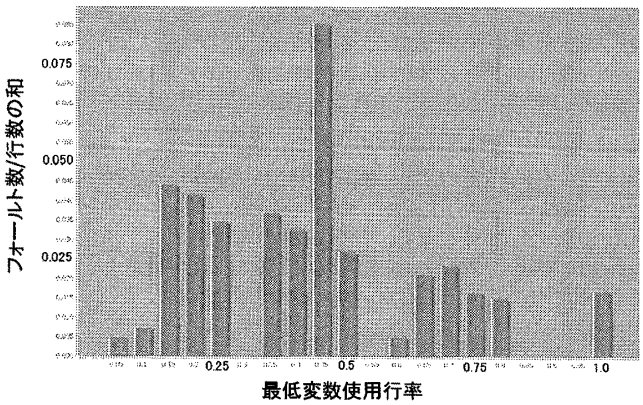


図 2: 最低変数使用行率と 1 行あたりのフォールト数

4 実験

提案手法の有効性を調査するため実験を行った．

4.1 実験対象

本実験では，Apache Ant(v1.1) と Ant のリポジトリよりメソッド単位で構築したフォールト情報を用いた．Apache Ant(v1.1) の規模を表 1 に示す．

フォールト情報はリポジトリの変更内容を見て，変更がフォールトの修正であるか，フォールトの修正ならばどのメソッドかを 1 つずつ確認し作成した．フォールト情報を構築するのに使用した Ant のバージョンは 1.1 から 1.7.1 であり，1,514 個のフォールトがメソッド単位で発見された．Ant 1.1 から Ant1.7.1 の開発期間は 7 年 11 カ月である．

4.2 実験方法

メソッドの行数が多いほどフォールトを含みやすい．そこでメソッド行数によるフォールトの公平さを取るため 1 行あたりのフォールト数と各属性値との関係をグラフ (図 2) にして調査した．また，各属性値とフォールト数・メソッド 1 行あたりのフォールト数について Spearman の順位相関係数も求めた．ここでの行数は，コメント・空行も含まれる．

4.3 実験結果

図 2 は最低変数使用行率と 1 行あたりのフォールト数の関係をグラフにしたものである．横軸は最低変数使用行率であり，0 から最大値までを 20 分割し，軸の数値で区切られた間の値を取るメソッドがその区間に含まれる．縦軸は，区間内の数値を取るメソッドの 1 行あたりのフォールト数であり，区間内のメソッドに含まれたフォールト数を区間内のメソッド行数の和で割ったものである．最低変数使用行率が小さいほど 1 行あたりのフォールト数が多い傾向にある．紙面の都合上図は省略しているが，平均変数使用行率・平均変数使用行差では，実際に使用されている範囲と使用可能な範囲の差が小さ

表 1: Apache Ant(v1.1) の規模

ソースファイル数	87
クラス数	112
行数	17,412
メソッド数	647

いものにフォールト数が多かった．しかし，変数使用行率の加重平均・最低値と変数使用行差の最大値では，2つの範囲の差が大きいほど 1 行あたりのフォールト数が多かった．

メソッドの各属性値とフォールト数・メソッド 1 行あたりのフォールト数の Spearman の相関係数を表 2 に示す．最低変数使用行率・平均変数使用行差・加重平均変数使用行差・最大変数使用行差の相関係数は，行数とサイクロマチック数の相関係数より絶対値が大きい．[*1]

順位相関係数は全て，1% 水準で有意であった．

5 考察

平均変数使用行率・平均変数使用行差では，実際に使用されている範囲と使用可能な範囲の差が小さいものにフォールト数が多かったが，変数使用行率の加重平均・最低値と変数使用行差の最大値は，実際に使用されている範囲と使用可能な範囲の差が大きいほど 1 行あたりのフォールト数が多かった．つまり，単純に平均をとったものではなく，変数使用行率が低い変数・変数使用行差が大きい変数が 1 つでも含まれるメソッドに 1 行あたりのフォールト数が多いといえる．

また表 2 より，最低変数使用行率・平均変数使用行差・加重平均変数使用行差・最大変数使用行差の相関係数は行数とサイクロマチック数の相関係数より絶対値が大きいので，よりフォールトとの相関があり，Fault-Prone メソッドの特定に有効であると考えられる．

計測した行数にはコメント・空行が含まれるため，コーディングスタイルの影響を受ける．しかし，コメントや空行が多く含まれる箇所は，処理が複雑であると予想される．そのような箇所でスコープが広いと，よりフォールトを生みやすい．そのため，計測した行数にコメント・空行が含まれることは，変数のスコープと実際に使用されている差による影響をより強調したことになる．

6 あとがき

本研究では，変数使用に着目した Fault-Prone メソッドを特定する手法を提案した．変数のスコープと変数の実際に使用されている範囲の差に着目し，変数使用行率・変数使用行差の有効性を調査するため実験を行った．実験の結果，変数使用行率が低い変数が 1 つでも含まれるメソッドに 1 行あたりのフォールト数が多いことが確認できた．また，最低変数使用行率・加重平均変数使用行差・最大変数使用行差の Spearman の順位相関係数は，行数とサイクロマチック数の順位相関係数より絶対値が大きく，よりフォールトとの相関があると確認した．

表 2: Spearman の順位相関係数

		フォールト数	1 行あたりのフォールト数
変数使用行率	平均	0.258	0.237
	加重平均	0.255	0.235
	最低値	-0.328	-0.301
変数使用行差	平均	0.335	0.307
	加重平均	0.340	0.312
	最大値	0.335	0.306
行数		0.320	0.292
サイクロマチック数		0.310	0.280

[*1] 変数使用行率は負の相関が強いほど良い結果である

7 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] T.McCabe. A software complexity measure. *IEEE Transactions on Software Engneering*, Vol. SE-2, pp. 308–320, 1976.

[2] 山田茂, 高橋宗雄. ソフトウェアマネジメントモデル入門－ソフトウェア品質の可視化と評価法. 共立出版, 1993.

[3] S.R.Chidamber and C.F.Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engneering*, Vol. 20, No. 6, pp. 476–493, 1994.

[4] V.R.Basili, L.C.Briand, and W.L.Melo. A validation of object oriented metrics as quality indicators. *IEEE Transactions on Software Engineering*, Vol. 22(10), pp. 751–761, 1996.

[5] P.L.Li, J.herbsleb, M.Shaw, and B.Robinson. Experiences and results from initiating field defect prediction and product test prioritization efforts at abb inc. *28th Int'l Conf. on Software Engineering*, pp. 413–422, 2006.

[6] E.Arisholm and L.C.Briand. Predicting fault-prone components in a java legacy system. *Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering*, pp. 8–17, 2006.

[7] JIS X 0129(ISO/IEC 9126). ソフトウェア製品の評価-品質特性及びその利用要領. 日本規格協会, 1994.

[8] 門田暁人, 佐藤慎一, 神谷年洋, 松本健一. コードクローンに基づくレガシーソフトウェアの品質の分析. 情報処理学会論文誌, Vol. 44, No. 8, pp. 2178–2187, 2003.

[9] 馬場慎太郎, 吉田則裕, 楠本真二, 井上克郎. Fault-Prone モジュール予測へのコードクローン情報の適用. 電子情報通信学会論文誌 D, Vol. 91, No. 10, pp. 2559–2561, 2008.