Fault-Prone モジュール予測へのコードクローン情報の適用

馬場慎太郎[†]　吉田　則裕[†]（学生員）
楠本　真二[†]（正員）　井上　克郎[†]（正員）

Application of Code Clone Information to Fault-Prone Module Prediction

Shintaro BABA[†], *Nonmember*,
Norihiro YOSHIDA[†], *Student Member*, Shinji KUSUMOTO[†], and Katsuro INOUE[†], *Members*

[†] 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, 560–8531 Japan

あらまし　本論文では，Fault-Prone モジュール予測にコードクローン情報を用いた実験結果について述べる．従来用いられてきた複雑度メトリックスのほかに，コードクローンに関するメトリックスも説明変数に加えたロジスティック回帰分析により Fault-Prone モジュールを予測した．その結果，従来手法に比べて予測精度が向上することを確認した．

キーワード　Fault-Prone，コードクローン，複雑度メトリックス，ロジスティック回帰

1. まえがき

高品質なソフトウェア作成のために，早期段階での不具合（フォールト）を含みやすいモジュールの特定が求められており，様々な研究が行われている [1], [3]．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ッ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ことが多い．

一方，近年ソフトウェアの保守性に影響を与える要因として，コードクローン（ソースコード中の類似したまたは同一のコード片の集合）が指摘されている．更に，コードクローンに関するメトリックスと複雑度メトリックスはそれぞれ保守性に関する異なる側面を評価しているという報告もあるが [2]，筆者らの知る限り，コードクローン情報を用いた Fault-Prone モジュールの予測手法は提案されていない．

そこで，本論文では複雑度メトリックスに加えてコードクローンに関するメトリックスも説明変数としたロジスティック回帰分析によって Fault-Prone モジュールの予測を試みた．対象としたのは複数社で共同開発された中規模情報システムである．

2. 実験方法

2.1 対象プロジェクト概要

対象プロジェクトは，経済産業省の支援を受けて，COSE(注1) 参加企業によって実施されたプローブ情報システムの開発プロジェクトである [5]．このプロジェクトには以下の特徴がある．

- 2 か年度にわたったプロジェクトであり，新規開発を行った 2005 年度版と，保守や改良を行った 2006 年度版のデータが取得可能である．
- 実装内容別に分類されたコンポーネントが存在する．コンポーネントは一つ以上のファイルの集合であり，すべてのファイルはいずれか一つのコンポーネントに属している．
- フォールト情報はコンポーネントごとに記録されている．したがって，これらの記録を調べることで，2005 年度，2006 年度に開発・保守されたモジュールごとのフォールトの有無が把握できる．

上記の特徴から，本論文ではコンポーネントをモジュールとみなす．つまり Fault-Prone か否かの予測はコンポーネントごとに行う．また，2005 年度に開発された 40 モジュールの情報から予測モデルを作成し，2006 年度に開発・保守された 32 モジュールに対して適用を行う．なお，これらのモジュールは C/C++で実装されたものである．

2.2 複雑度メトリックス

計測可能な範囲でできるだけ一般的なメトリックスを選定した．具体的には次の五つである．

- 関数ごとのサイクロマチック数
- ファイルごとのコード行数
- ファイルごとのコメント率
- 関数ごとの IFANIN（入力となるパラメータとグローバル変数の数）
- 関数ごとの IFANOUT（出力するパラメータとグローバル変数の数）

また，本データではおよそ 8 割のモジュールがフォールトを含んでいるという特徴がある．このような場合，一つでも複雑なファイルや関数があれば，そこにフォールトがある可能性が高いと思われる．よって，モジュールのメトリックス値は，そのモジュール内のファイルや関数等ごとの計測値の最大値とした．なお，計測には Understand for C++ [6] を用いた．

(注1)：ソフトウェアエンジニアリング技術研究組合（COSE）：COnsortium for Software Engineering.

2.3 コードクローンに関するメトリックス

コードクローン分析ツール Gemini を用いてモジュールごとに以下の二つのメトリックスを計測した．

- NOC（Number Of Clone）

モジュール中のコードクローンとなっているコード片の個数．

- ROC（Ratio Of Clone）

$$ROC = \frac{\text{コードクローンとなっているトークン数}}{\text{（同一組織が作成した）モジュール内のすべてのトークン数}} \times 100(\%)$$

また，コードクローン自身を特徴づけるメトリックスとして RNR（Ratio of Non-Repeated code）がある [4]．RNR は，トークンベースでそのコードクローンの非繰返し度を表す．この値が 50 未満の繰返し構造が多いコードクローンは保守性にあまり影響を与えないといわれているため，RNR によるフィルタリングをした方が精度が向上する可能性がある．そこで本論文では，RNR 値 50 以上のコードクローンしかコードクローンとみなさない，RNR フィルタリングを行う場合と，行わない場合，両方について適用を行った．

3. 実験結果

表 1 は得られた実験結果の凡例である．N_1〜N_4 には該当するモジュール数が入る．例えば N_2 は，予測ではフォールトを含む（F）と判定したが実際にはフォールトを含まなかった（NF）(注2) モジュール数となる．本論文では結果を評価するための指標として，再現率と適合率を用いるが，それぞれ以下の式で表される．ともに予測精度が高いほど大きな値となる．

$$\text{再現率} = \frac{N_4}{N_3 + N_4}, \quad \text{適合率} = \frac{N_4}{N_2 + N_4}$$

実験結果と再現率，適合率を表 2 に示す．コードクローンに関するメトリックス(注3) を説明変数に加えることでおおむね予測精度が向上している．しかし予測精度が低下した場合もあり，RNR フィルタリングを行わなかった場合では全体的に精度向上が確認できなかった．このことから，RNR 値が低いコードクローンは，Fault-Prone モジュールの予測に関して有用ではないことが分かる．RNR 値が低いコードクローンは変数宣言部等に多く存在するが，そのような単純な部分ではあまりフォールトが作り込まれないために，このような結果になったと考えられる．

表 1 実験結果の凡例
Table 1 Explanatory notes to experimental results.

		予測	
		NF	F
実測	NF	N_1	N_2
	F	N_3	N_4

4. むすび

本論文では，コードクローンに関するメトリックスを用いた Fault-Prone モジュールの予測の実験を行った．その結果，RNR メトリックスによってフィルタリングを行えば，従来の手法に比べて予測精度が向上することを確認した．

今後は多様なプロジェクトに対して適用を試み，プロジェクトの特性がどのように結果に反映されるか確認していきたいと考えている．

表 2 実験結果
Table 2 Experimental results.

複雑度メトリックスのみ（従来手法）		予測 NF	予測 F	
実測	NF	2	6	再現率：0.875
	F	3	21	適合率：0.778

RNR フィルタリングありで ROC と NOC を追加		予測 NF	予測 F	
実測	NF	4	4	再現率：1.000
	F	0	24	適合率：0.875

RNR フィルタリングありで ROC を追加		予測 NF	予測 F	
実測	NF	2	6	再現率：0.917
	F	2	22	適合率：0.786

RNR フィルタリングありで NOC を追加		予測 NF	予測 F	
実測	NF	4	4	再現率：1.000
	F	0	24	適合率：0.875

RNR フィルタリングなしで ROC と NOC を追加		NF	F	
実測	NF	2	6	再現率：0.833
	F	4	20	適合率：0.769

RNR フィルタリングなしで ROC を追加		予測 NF	予測 F	
実測	NF	2	6	再現率：0.958
	F	1	23	適合率：0.793

RNR フィルタリングなしで NOC を追加		予測 NF	予測 F	
実測	NF	2	6	再現率：0.875
	F	3	21	適合率：0.778

(注2)：あくまで，2006 年度の開発・保守期間にフォールトが発見されなかったという意味である．
(注3)：ROC の値は，RNR フィルタリングありの場合，40.4（2005 年度）と 39.7（2006 年度），RNR フィルタリングなしの場合，48.1（2005 年度），46.6（2006 年度）（いずれも平均値）となった．

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．また，日本学術振興会科学研究費補助金基盤研究（A）（課題番号：17200001）の助成を得た．

文　献

[1] E. Arisholm and L.C. Briand, “Predicting fault-prone components in a Java legacy system,” Proc. ISESE2006, pp.8–17, Rio de Janeiro, Brazil, Sept. 2006.

[2] 馬場慎太郎，吉田則裕，楠本真二，井上克郎，“ソフトウェア保守性を評価するメトリクス間の関連分析,” ソフトウェア信頼性研究会第 4 回ワークショップ論文集，pp.101–106, June 2007.

[3] V.R. Basili, L.C. Briand, and W.L. Melo, “A validation of object oriented metrics as quality indicators,” IEEE Trans. Softw. Eng., vol.22, no.10, pp.751–761, Oct. 1996.

[4] 肥後芳樹，吉田則裕，楠本真二，井上克郎，“産学連携に基づいたコードクローン可視化手法の改良と実装,” 情処学論，vol.48, no.2, pp.811–822, Feb. 2007.

[5] 情報処理推進機構ソフトウェア・エンジニアリング・センター：ソフトウェアエンジニアリングの実践—先進ソフトウェア開発プロジェクトの記録，翔泳社，東京，2007.

[6] http://www.scitools.com/products/understand/cpp/product.php

（平成 20 年 4 月 4 日受付，5 月 29 日再受付）