情報処理学会論文誌 Vol.53 No.6 1635–1643 (June 2012)

開発履歴メトリクスを用いた細粒度なFault-proneモジュール予測

畑 秀明[1,a)] 水野 修[2] 菊野 亨[1]

受付日 2011年10月9日, 採録日 2012年3月2日

概要：Fault-proneモジュールの予測において，ソフトウェアリポジトリから収集可能な開発履歴メトリクスは有用であることが数多くの文献で報告されている．開発履歴メトリクスには，コードに関するもの，プロセスに関するもの，開発組織に関するものなどがある．これらのメトリクスの収集はファイルレベルでは容易であるが，より細粒度なメソッドレベルでは収集が困難であった．これは，版管理システムがソースコードをファイルレベルで管理するためである．本稿では，以前に提案した細粒度履歴管理リポジトリを用いることでメソッドレベルの開発履歴メトリクスの収集を行う．オープンソースソフトウェアプロジェクトを対象に開発履歴メトリクスを細粒度モジュール（メソッドレベル）とファイルレベルで収集し，Fault-proneモジュール予測を行った．工数を考慮した評価から，細粒度モジュールでのFault-proneモジュール予測がファイルレベルに比べて有用であることを確認した．

キーワード：Fault-proneモジュール予測，細粒度予測，細粒度マイニング，開発履歴メトリクス，工数を考慮した評価

Fault-prone Module Prediction on Fine-grained Modules Based on Historical Metrics

Hideaki Hata[1,a)] Osamu Mizuno[2] Tohru Kikuno[1]

Received: October 9, 2011, Accepted: March 2, 2012

Abstract: Many studies reported that historical metrics collected from software repositories are useful for fault-prone module prediction. There are many historical metrics proposed in literature, such as code-related, process-related, and organization-related metrics. Since source code management system stored file-level histories, it has been difficult to collect historical metrics of fine-grained modules compared to file-level historical metrics. Using our fine-grained version control system, this paper conducts a comparative study of fault-prone module prediction on a file-level and a method-level. We empirically evaluated our prediction models with open source software projects. Based on effort-aware models, fault-prone module prediction models on fine-grained modules perform better than file-level models.

Keywords: fault-prone module prediction, fine-grained prediction, fine-grained mining, historical metrics, effort-based evaluation

1. はじめに

版管理システムや障害管理システムといったソフトウェアリポジトリは，実際のソフトウェア開発履歴を蓄積しており，実際に何が起こったか，どのようなパターンがあるかといった，プロジェクト特有の有用なデータが豊富に含まれていると考えられる．このような観点から，ソフトウェアリポジトリからのデータマイニングは近年活発に行われている [34]．

同様のデータマイニングは，Fault-proneモジュール予測の研究においても注目されている．ソフトウェアリポジトリから収集可能な開発履歴に関するメトリクスがいくつか提案され，多くの研究でFault-proneモジュールの予測

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[2] 京都工芸繊維大学大学院工芸科学研究科
Graduate School of Science and Technology, Kyoto Institute of Technology, Kyoto 606–8585, Japan
[a)] h-hata@ist.osaka-u.ac.jp

モデル構築に用いられている．文献 [5] では，近年の研究成果をもとに Microsoft 社内で CRANE と呼ばれるシステムを構築し，実際に Windows Vista の開発とメンテナンスで活用した結果が報告されている．このシステムは，開発履歴メトリクスを用いてリリース後の障害を予測し，変更分析と組み合わせてテストの優先付けを行う．メンテナンスの修正プロセスにおいて，早いフェーズから修正の配布までの各フェーズでリスク分析のサポートをする様子が紹介され，その有用性が主張されている．

データマイニングの対象となるソフトウェアリポジトリの代表的なものには，版管理システムがある．版管理システムには，ソースコードに対する変更行や変更回数，変更を行った時刻や開発者の名前などが記録されているため，そのソースコードに対して，コードに関するメトリクス，プロセスに関するメトリクス，開発組織に関するメトリクスなどが収集可能である．本稿では，これらの開発履歴から収集可能なメトリクスを開発履歴メトリクスと呼ぶ．版管理システムでは履歴の情報はファイルレベルで蓄積されるため，これまでは，収集する開発履歴メトリクスもファイルレベルであり，これを用いた Fault-prone モジュールの予測もファイルかそれより大きいモジュールレベルであった．

予測対象となるモジュールの粒度は，Fault の発見と修正プロセスの工数に関わる．この工数を考慮した予測結果の評価法が，近年提案されている [2], [17]．これは，予測結果を活用した，開発や保守の工程を実施する場合の工数を考慮するもので，実用上有用と考えられる．Kamei らは，ファイルレベルとパッケージレベルという異なるモジュール粒度での Fault-prone モジュール予測について工数を考慮した評価から，粒度の小さいファイルレベルの方が良い結果を得たことを報告している [12]．これは，あるモジュールに対して正しく Fault の潜在を予測できても，モジュールの粒度の大きさが大きいほど，そのモジュールの Fault 発見への工数は大きくなるだろうという直観にも合う．

このように工数を考慮すると，細粒度なメソッドレベルでの Fault-prone モジュール予測は好ましいのではないかと考えられるが，ファイルレベルと同様な開発履歴メトリクスの収集は困難であり，同様のアプローチでの細粒度モジュールの Fault-prone モジュール予測はこれまで十分行われていない．

本稿では，先に提案した細粒度履歴管理リポジトリ [11], [33] を用いて，Java のメソッドに対する開発履歴メトリクスを収集し，Fault-prone なメソッドの予測を行う．我々は同様の試みの初期段階を文献 [10] で報告している．本稿は，収集するメトリクスと予測結果の評価法について，より詳細な議論を行う．収集する開発履歴メトリクスは，コードに関するメトリクス，プロセスに関するメトリクス，開発組織に関するメトリクスなどである．Eclipse に関する 4 つのオープンソースソフトウェアプロジェクトに対して，ファイルレベルとメソッドレベルの Fault-prone モジュール予測を行った．工数を考慮した評価から，メソッドレベルでの予測の方がファイルレベルと比べて良い結果が得られた．

以降，2 章で開発履歴メトリクスを紹介し，3 章で実験方法を説明する．4 章で実証的な実験の結果を報告し，5 章で結果について議論する．最後に 6 章で本稿をまとめる．

2. 開発履歴メトリクス

不具合の表記には，Fault，Defect，Bug，Failure などがある．主に研究されている不具合は以下のように分類できる．

Fault：不具合を引き起こすソースコードの一部．修正されることによって分かる．Defect，Bug と表記されることもある．

障害：Failure．リリース後に報告されることによって分かる不具合．Fault に起因することから，障害の予測はリリース後に残った Fault の予測となる．

ビルドエラー：ビルドの失敗．

本章では，これらの不具合の違いを区別せず，開発履歴メトリクスとして提案，利用されているメトリクスに関する研究を調査する．開発履歴メトリクスは計測対象ごとに，コード，プロセス，開発組織に関連するものに分類した．

2.1 コード関連

Nagappan らは，ソースコードの変更行数に関連したメトリクスで予測を行っている [21]．基本的なメトリクスとして，*Churned LOC*（最初のバージョンからの追加・修正行数の合計），*Deleted LOC*（最初のバージョンからの削除行数の合計），*Total LOC*（対象バージョンの行数）などを計測し，*ChurnedLOC*/*TotalLOC* や *DeletedLOC*/*TotalLOC* の値を説明変数とした Fault 密度予測モデルを構築し，Windows Server 2003 でのケーススタディから有用性を報告している．変更行数に関連したメトリクスは，以降も基本的なメトリクスとして多くの研究で利用されている [6], [12], [18], [20], [21], [26], [32]．

2.2 プロセス関連

本稿では，モジュールの変更回数や Fault の修正回数などの開発プロセスを対象とするメトリクスをプロセス関連の開発履歴メトリクスとしてまとめる．

初期の研究には，Graves らの成果があげられる．Graves らは，変更回数，過去の Fault 数，モジュールの存在期間などを計測して Fault の予測モデルを構築している [8]．電話システムを対象としたケーススタディから，これらのメトリクスが従来の複雑度メトリクスと比べて Fault 予測に

より有効であることを報告している．

変更回数 [6], [8], [9], [12], [14], [18], [20], [22], [23], [24], [29]，過去の Fault 数 [7], [8], [23], [32]，モジュールの存在期間 [6], [8], [9], [12], [14], [23], [26], [29]，なども多くの研究で利用されるメトリクスとなっている．また，Fault 修正の変更回数 [6], [9], [12], [14], [15], [20], [29]，Fault が混入されたことがあるモジュールと同時に変更される回数（ロジカルカップリング）[14], [19], [20] なども頻繁に用いられるメトリクスである．

2.3 開発組織関連

Graves らは，開発した組織，開発者数などの開発組織に関連したメトリクスも計測している [8]．近年，企業データを対象とした研究で，開発組織に関連したメトリクスの有用性が多数報告されている．

Nagappan らは，「ソフトウェアの構造は開発する組織を反映する」という Conway の法則 [4] を実証的に分析するため，開発者数，脱退した開発者数，携わった組織数，オーナシップの組織階層，組織の凝集度，開発者の凝集度，組織の編集割合，といった組織メトリクスを提案している [22]．Windows Vista を対象とした適用実験から，変更行数メトリクスや複雑度メトリクスと比べても組織メトリクスは，障害予測において高い予測精度が得られることを報告している．

また，コードや開発者間のネットワークに関するメトリクスの適用が，複数の文献で提案されている [18], [24], [30]．彼らは，ネットワーク分析のメトリクスである，中心性や接続性，近接性メトリクスを計測し，リリース後の障害予測 [18], [24] やビルドエラーの予測 [30] を行っている．

2.4 従来の複雑度メトリクスとの比較

開発履歴メトリクスと従来の複雑度メトリクスとの比較調査も行われている [12], [20]．いずれの文献も変更回数や変更行数といった履歴情報に基づくメトリクスが，従来の複雑度メトリクスと比べて有用であると報告している．

3. 実験方法

3.1 細粒度モジュールの開発履歴メトリクス計測

本稿では，Java 言語で開発されたソフトウェアを対象とする．我々は，既存のファイルレベルの版管理システムの情報から，細粒度モジュールの履歴情報を再構築する細粒度履歴管理リポジトリを提案している [11], [33]．これは Git 上に構築するリポジトリで，各メソッドをそれぞれファイルとして保存することで版管理を可能にする．またファイル内容の類似度から履歴の追跡を行う．本システムにより，細粒度な Java のメソッドの履歴の追跡が可能となる．またメソッドシグネチャの変更があっても，ソースコードの行単位の類似度が十分大きい場合は対応付けを行い，追跡可能であることを実証的に示している．これを用いることで，ファイルレベルと同様の開発履歴メトリクスが，細粒度モジュール（メソッド）においても計測可能となる．

2 章を参考に，開発履歴メトリクスを計測する．本実験で計測した開発履歴メトリクスを**表 1** にまとめる．これらは多くの関連論文で計測される開発履歴メトリクスである．開発履歴メトリクスと複雑度メトリクスとの比較を目的とした研究でも同様の開発履歴メトリクスが計測されて

表 1 計測する開発履歴メトリクス
Table 1 Collected historical metrics.

分類	メトリクス	説明	計測している研究
コード関連	LOC	対象版の行数（開発履歴メトリクスではないが計測している）	文献 [12], [15], [21], [23], [26], [29], [32]
	AddLOC	最初の版からの追加行数	文献 [6], [12], [18], [20], [21], [26], [32]
	DelLOC	最初の版からの削除行数	文献 [6], [12], [18], [20], [21], [26], [32]
	AddPerLOC	AddLOC / LOC	文献 [21], [26], [32]
	DelPerLOC	DelLOC / LOC	文献 [21], [26], [32]
プロセス関連	ComNum	変更回数	文献 [6], [8], [9], [12], [14], [18], [20], [22], [23], [24], [29]
	FixComNum	Fault 修正の変更回数	文献 [6], [9], [12], [14], [15], [20], [29]
	FaultNum	過去の Fault 数（関連する障害レポート数）	文献 [7], [8], [23], [32]
	Period	存在期間（週単位）	文献 [6], [8], [9], [12], [14], [23], [26], [29]
	AvgPeriod	Period / ComNum	
	MaxInterval	変更間の間隔の最大期間（週単位）	
	MinInterval	変更間の間隔の最小期間（週単位）	
	LogCoupNum	Fault が発見されたモジュールと同時に変更された回数	文献 [14], [19], [20]
	BugIntroTiming	他のモジュールに Fault が混入されるタイミングで変更された回数	
開発組織関連	AuthorNum	そのモジュールを変更した開発者数	文献 [6], [8], [18], [20], [22], [24], [29], [32]

いる [12], [20]．コードに関するもの（2.1 節）とプロセスに関する主なメトリクス（2.2 節）を計測する．プロセス関連メトリクスの Fault に関する情報は，3.2 節で説明する SZZ アルゴリズム [27] で取得する．オープンソースソフトウェアプロジェクトにおいて，開発組織や組織のネットワークの情報を得ることは容易でないため，開発組織関連のメトリクス（2.3 節）としては，開発者数のみを計測する．

表 1 には，それぞれのメトリクスを計測している研究もまとめた．AvgPeriod，MaxInterval，MinInterval のメトリクスを計測している研究はない．本稿では，変更間の間隔に着目して計測を行う．また，BugIntroTiming も計測している研究はない．LogCoupNum は，変更の中で Fault が発見された特定のモジュールとの同時変更の回数に着目している．一方 BugIntroTiming は，変更の中でいずれかのモジュールに Fault が混入されるタイミングでの同時変更の回数に着目している．

3.2 Fault 情報の取得

オープンソースソフトウェアプロジェクトのリポジトリにおいて，いつどのモジュールに Fault が混入したかを特定するアルゴリズムとして，SZZ アルゴリズム [27] が広く使われている．版管理システムと障害管理システムの情報を用いて，障害管理システムに登録された障害の原因となった Fault の情報を取得する．SZZ アルゴリズムは，以下の 3 つのステップで特定を行う．

1. **Fault を修正した変更の特定**　登録された障害に関する Fault 修正の変更を探す．版管理システムのコミットログに障害レポートの ID 番号が記述されている変更を見つける．
2. **Fault を混入した変更の特定**　Fault 修正の変更時に編集された行が，その Fault に関わりが深いという仮定のもと，その行が最初に作成された変更を探す．具体的には，まず Fault 修正の変更で作成された版とその 1 つ前の版の間で `diff` を実行し，変更または削除された行を特定する．版管理システムの `annotate` や `blame` といったコマンドで，先ほど特定した行が最初に作成された変更を見つける．
3. **誤特定の除去**　先の 2 つの特定結果から不適切なものを取り除く．まず Fault 修正の変更が行われる日付は，対応する障害レポートが最初に報告された日付より後であり，またその障害レポートのステータスが修正済みに変わる前である場合のみが妥当と考えられる．そのため，その範囲にない変更は，その障害レポートに対する Fault 修正という特定は誤りであるとして除去する．次に Fault 混入の変更が行われる日付は，対応する障害レポートが最初に報告された日付より前である必要がある．そのため，ステップ 2 で起源を調査した行のうち，障害レポート報告日の前に作成された行のみが Fault 混入に関わりがある行と考えられる．そういった行が 1 行もないファイルは，その障害に関する Fault を含まないと判断する．また Fault を含むファイルが 1 つもない場合は，対応する Fault 修正の変更の特定が誤りだと判断する．

ステップ 2 における行の調査においては，Fault と関わりのない行の変更を含めないように，文献 [13] で行われているように，空行やコメントの編集やフォーマットの変更は無視することにした．SZZ アルゴリズムで特定した Fault 混入と修正の間の版を Fault ありとして以降の実験を行う．

3.3 予測モデルと工数を考慮した評価法

予測モデルの構築と予測精度評価には R [28] を用いる．予測モデルには，Random Forest [16] を採用し，R の *randomForest* パッケージを用いた．文献 [12], [17] などでも同じパッケージが用いられている．

10 分割の交差検証により予測の評価を行う．本稿では粒度の異なる Fault-prone モジュール予測の結果について工数を考慮して評価するため，同様の試みを行った先行研究 [2], [12], [17], [25] を参考にする．これは，テストやレビューの工数はほぼモジュールのサイズに比例するという知見 [2] からモジュールの行数（LOC）を工数とする評価法である．

Fault-prone モジュール予測で算出した，Fault-prone である確率の大きい順にモジュールを調査する状況を考える．予測結果を活用した開発や保守の工程まで考えると，工数の増大を抑えつつ多くの Fault を発見したい．そこで，調査モジュールの行数が全体行数に対して一定の割合のとき，全体の中でどれだけの Fault を含むモジュールを発見できるか（カバー率）で評価する．文献 [12], [25] にならい，一定の割合を 20%とする．すなわち，調査したモジュールの累積行数が全体の 20%に達したときのカバー率で比較する．

先行研究のうち文献 [12], [17], [25] は Fault 密度（行数あたりの Fault 数）を，文献 [2] は Fault の有無を計測して評価を行っている．オープンソースソフトウェアを対象とした場合，モジュール中の Fault 数の計測には，対応する障害レポートを数えあげることが考えられるが，障害レポート全体には数十%も重複があることが報告されている [3]．このため Fault 数を正しく計測することは困難であると考え，本稿では Fault の有無のみに着目する．

3.4 対象プロジェクト

実験対象には，オープンソースソフトウェア Eclipse のサブプロジェクトである Xpand（xpd）[*1], Webtools Incubator（wti）[*2]，EMF Compare（emfc）[*3]，Eclipse Communica-

[*1] http://git.eclipse.org/c/m2t/org.eclipse.xpand.git/
[*2] http://www.eclipse.org/webtools/incubator/
[*3] http://git.eclipse.org/c/emfcompare/org.eclipse.emf.compare.git/

表 3 交差検証対象モジュールデータ
Table 3 Target module data.

プロジェクト	対象リビジョン		ファイル		メソッド	
	タグ	作成日	Fault あり	Fault なし	Fault あり	Fault なし
xpd	Galileo_RC1	2009-05-18	115	1,132	295	8,248
wti	v20090510	2009-05-10	140	466	317	5,175
emfc	R0_8_0	2008-08-26	177	162	424	2,079
ecf	Root_Release_3_0	2009-06-02	200	1,515	619	10,502

表 2 実験対象プロジェクトデータ
Table 2 Target project data.

プロジェクト	開始	最新	変更回数（コミット数）
xpd	2007-11-10	2011-05-11	1,017
wti	2007-11-10	2010-07-22	1,133
emfc	2007-04-03	2011-05-05	1,587
ecf	2004-12-03	2011-05-13	9,742

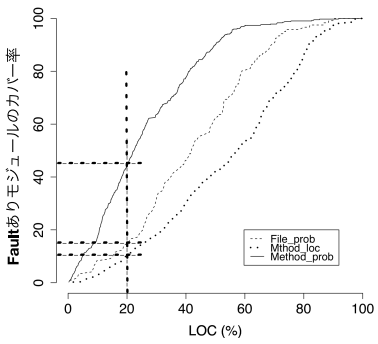
tion Framework（ecf）[*4]を選択した．これらのプロジェクトは Java 言語で記述されており，版管理システム Git で管理されている．2011 年 5 月 14 日に Git リポジトリのクローンを取得した．**表 2** に各プロジェクトの開始日，最新の変更日，変更回数の情報を示す．また障害管理システム Bugzilla [*5]から 2011 年 4 月 30 日までの障害レポートデータを取得した．

交差検証を行うモジュールの情報を**表 3** に示す．表 3 に示したタグが付けられたリビジョンのモジュールを対象にした．予測はファイルレベルとメソッドレベルで行う．Fault 有無のモジュール数をファイルレベルとメソッドレベルでそれぞれ示す．これらは，ファイルレベルとメソッドレベルにおいて，3.2 節の SZZ アルゴリズムで Fault ありモジュールとされたものである．あるリビジョンでの Fault 有無の決定は次のように行う．各モジュールごとに Fault が含まれている期間（混入された版から修正される直前の版まで）が，SZZ アルゴリズムによって特定できる．そこで，対象リビジョンの版が Fault を含む期間に含まれているモジュールを Fault ありモジュールと決定する．

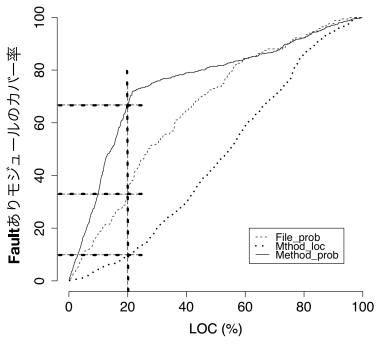
4. 結果

3.3 節で述べたように，調査する行数を工数と考える．ここで，調査対象の全体の行数は全メソッドの累積行数とする．すなわちファイルレベルであっても調査対象はメソッド部分のみとする．

図 1 に，`emfc` プロジェクトと `ecf` プロジェクトで，モジュールを順に調査した場合の累積調査行数と Fault ありモジュールのカバー率の推移をまとめたグラフを示す．モジュールは，Fault-prone な確率が大きい順に調査する．また，予測モデルの有用性を確認するために，単純にメソッドの LOC が大きい順に調査した場合の結果もあわせて示す（間隔の広い点線）．



(a) `emfc` プロジェクト



(b) `ecf` プロジェクト

図 1 ファイルレベルとメソッドレベルにおける累積行数と Recall の推移
Fig. 1 LOC-based cumulative lift chart for file-level v.s. method-level.

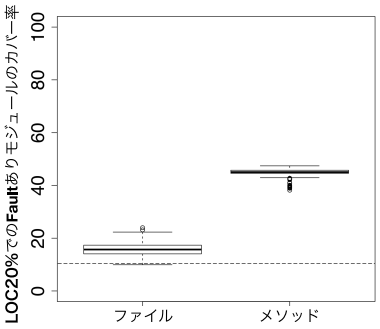
調査行数が全体の 20%となるときの Fault ありモジュールのカバー率を点線で示す．図 1 から，メソッドレベルの予測結果はファイルレベルの予測結果と比べて，同程度の調査行数で高いカバー率となり，また少ない調査行数で同程度のカバー率を得ることが確認できる．メソッドの LOC 降順に調査した結果は，同程度の調査行数での Fault ありモジュールのカバー率が最も低い．こうした結果は，他のすべてのプロジェクトでも得られた．

図 1 は 1 回の試行の結果を図示したものであった．すなわち予測モデルを 1 回構築して 10 分割の交差検証を行った結果である．本稿で予測モデルとして採用した Random Forest は乱数的アルゴリズムである．乱数的アルゴリズムは，結果を正しく評価するため 1,000 回ほどのランダム試行

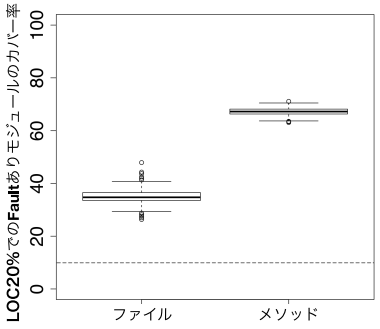
[*4] http://git.eclipse.org/c/ecf/org.eclipse.ecf.git/
[*5] https://bugs.eclipse.org/bugs/

表 4 総行数 20%での Fault ありモジュールのカバー率（1,000 回試行の中央値）
Table 4 Ratio of faulty modules on 20% LOC (median in 1,000 models).

プロジェクト	メソッド LOC 降順	ファイルレベル予測 (a)	メソッドレベル予測 (b)	(b) − (a)
xpd	6.4	9.3	41.4	32.1
wti	2.5	38.2	60.3	22.1
emfc	10.4	15.7	45.1	29.4
ecf	9.9	34.7	67.2	32.5



(a) emfc プロジェクト



(b) ecf プロジェクト

図 2 総行数の 20%での Fault ありモジュールのカバー率中央値の箱ひげ図（1,000 回試行）
Fig. 2 Boxplot of faulty module ratio on 20% LOC (median in 1,000 models).

を実施することが推奨されている [1]．**図 2** に，emfc プロジェクトと ecf プロジェクトでの，1,000 回試行の結果を示す．図 2 は，1,000 回試行して得られた総行数の 20%での Fault ありモジュールのカバー率の箱ひげ図である．メソッドの LOC 降順に調査した結果を点線で示す．1,000 回の試行から，総行数 20%でファイルレベルと比べてメソッドレベルの予測結果は高いカバー率であることが明確に確認できる．また LOC 降順に調査した結果と比べても，メソッドレベルの予測結果が高いカバー率であることが分かる．他のプロジェクトでも，同様の結果が得られた．

総行数 20%での Fault ありモジュールのカバー率の 1,000 回試行での中央値を，**表 4** にファイルレベルとメソッドレベルでまとめた．また，メソッドの LOC 降順の結果も示す．すべてのプロジェクトで，メソッドの LOC 降順のカバー率が最も低く，予測結果では，メソッドレベルがファイルレベルに比べて高いカバー率を得た．すなわち，同程度の工数でメソッドレベルの予測結果を用いたものが Fault ありモジュールをより多く調査できるといえる．表 4 の 5 列目は，メソッドレベルとファイルレベルでの予測結果のカバー率の差である．差分から，ファイルレベルと比べてメソッドレベルはカバー率が 22%から 32%の間で向上していることが分かる．メソッドレベルの予測では，総行数の 20%までに Fault ありモジュールの 40%以上が含まれている．以上の結果から，メソッドレベルでの Fault-prone モジュール予測がファイルレベルと比べて，工数を考慮すると有用であることが確認できた．

表 5 Random Forest モデルにおける開発履歴メトリクス重要度の順位
Table 5 Variable importance rank measured by a Random Forest.

		xpd		wti		emfc		ecf	
メトリクス	順位中央値	F.	M.	F.	M.	F.	M.	F.	M.
LOC	1	1	1	1	1	5	1	1	1
MaxInterval	3	5	2	2	2	10	3	3	3
AvgPeriod	3.5	9	7	3	5	2	4	2	2
AddLOC	4	3	5	4	4	3	2	6	6
Period	4.5	4	6	6	3	7	5	4	4
MinInterval	5.5	6	3	9	6	4	7	5	5
LogCoupNum	7	8	11	7	7	6	6	7	7
DelLOC	8.5	2	4	5	8	11	12	9	10
ComNum	8.5	11	10	8	9	1	9	8	8
BugIntroTiming	10	10	9	10	10	9	10	10	9
FixComNum	11	13	12	11	11	12	8	11	11
AuthorNum	12	7	8	12	13	8	13	13	12
AddPerLOC	13.5	12	13	13	12	14	14	14	14
FaultNum	13.5	14	15	14	14	13	11	12	13
DelPerLOC	15	15	14	15	15	15	15	15	15

F.：ファイルレベル，M.：メソッドレベル

5. 議論

5.1 開発履歴メトリクスの重要度

予測モデルの構築に用いた開発履歴メトリクスの中で，どのメトリクスが有用であったか，またファイルレベルとメソッドレベルで重要となるメトリクスに違いが見られるかについて議論する．

表 5 に，Random Forest モデル構築における重要度の順位を，各開発メトリクスに対してまとめた．重要度の順位は *randomForest* パッケージで取得できる．それぞれのプロジェクトにおいて，ファイルレベルとメソッドレベルで

構築された Random Forest モデルから重要度の順位を取得している．表 5 では，各開発メトリクスにおける順位の中央値が降順になるようまとめている．上位のメトリクスほど多くの予測モデルで重要度が高かったと考えられる．

重要度が最も高いメトリクスは LOC であった．他の上位には，期間に関連した MaxInterval，AvgPeriod，Period，MinInterval や，コード関連の AddLOC，DelLOC が見られる．一方，コード関連の AddPerLOC，DelPerLOC や，過去の Fault に関する FaultNum，FixComNum や編集人数 AuthorNum などは下位に見られる．

プロジェクトごとに見ると，`wti` と `ecf` ではファイルレベルとメソッドレベルで，メトリクスの順位に大きな違いは見られない．一方，`xpd` と `emfc` プロジェクトでは，メトリクスの上位の順位に違いが見られる．ただし下位に大きな違いは見られない．特に `emfc` プロジェクトのファイルレベルにおけるメトリクスの順位は他のケースとの違いが大きい．表 3 を見ると，`emfc` プロジェクトのファイルレベルにおいてのみ，全モジュール中の Fault ありモジュールが半分以上になっていて，全体に占める割合がとても大きいことが分かる．このため，他のケースでは重要度が高くなかったメトリクスも，本ケースではモデル構築に役立ったのではないかと考えられる．メソッドレベルでもファイルレベルと同様な開発履歴メトリクスが予測モデル構築に有効といえるが，詳細な分析は今後の課題である．

本結果から，重要度の高かった開発履歴メトリクスは，ファイルレベルと同様にメソッドレベルでも計測することで，有効に活用できることが期待できる．一方，本実験で重要度の低かった開発履歴メトリクスは予測モデルの構築に十分寄与しないことが多いかもしれない．開発履歴メトリクスの計測のコストが大きい場合には，本結果で得られた重要度の高いものから計測すると効果が高いと思われる．しかし，プロジェクトによっては異なる結果が得られることもあるため，できるだけ多くの開発履歴メトリクスを計測することを推奨する．

5.2 妥当性への脅威

本稿で Fault ありの有無を特定するのに採用した SZZ アルゴリズは，障害レポートの ID 番号とコミットログをもとに関連する情報を特定する．そのため，障害レポートやコミットログに明記されていない場合は関連する Fault 情報の特定ができない．また誤特定も完全には除去できない [27]．SZZ アルゴリズムは多くの関連研究で用いられているが，こうした誤特定や見逃しがあることが知られており，本稿の実験でも誤った Fault 情報が含まれている恐れがある．これらの問題に対して，Fault と変更の情報をリンクづけする精度を高めた新たな手法が提案されている [31]．この新しい手法を採用することで，誤特定や見逃しを少なくすることが期待できる．

評価実験は，統合開発環境の Eclipse に関連したプロジェクトのみを対象としたものである．4 つすべてのプロジェクトで有用性が確認できたが，一般性を議論するためには，特に異なるドメインのプロジェクトへの適用が必要である．また，本稿で対象とした言語は Java に限定している．他の言語のソフトウェアでも同様の結果が得られるかどうかの調査が必要である．さらに，対象としたプロジェクトはすべてオープンソースソフトウェアプロジェクトである．商用のソフトウェアプロジェクトでは，異なる開発プロセスから異なる結果が得られる可能性はある．

今回対象としたプロジェクトの特徴から適用可能性を考える．4 つのプロジェクトは，開発者数は数人から数十人程度の小規模であるが，開発期間は 3 年以上あり開発履歴は蓄積されている．変更回数も 1,000 回を超えており，有用な開発履歴メトリクスが収集できたと思われる．開発履歴が十分に蓄積されないと適切な開発履歴メトリクスを収集できないので，開発期間の短いプロジェクトには適用が難しい場合があるかもしれない．

5.3 メソッドレベルの Fault-prone モジュール予測のデメリット

メソッドレベルの Fault-prone モジュール予測は，ファイルレベルと比べて同工数で Fault ありメソッドを多く調査することができることを示せた．しかし，フィールドの初期値の変更などメソッド以外の Fault を予測対象とできない．

また，本稿で構築した Fault-prone モジュール予測器は，モジュール単体が Fault-prone かどうかを予測する．したがって，モジュール間の依存関係の Fault はうまくモデル化できていない．メソッド間の依存関係を考慮したモデル化が必要となる．

6. まとめ

本稿では，これまで研究されてきたファイルレベルの開発履歴メトリクスを，細粒度なメソッドレベルで計測し，Fault-prone モジュール予測に適用した結果を報告した．4 つのオープンソースソフトウェアプロジェクトを対象とした実証的な実験で，工数を考慮した評価からメソッドレベルでの Fault-prone モジュール予測がファイルレベルと比べて有用であることを確認した．

Fault-prone モジュール予測に用いたツールや手法は多くの文献で用いられているものである．また必要なデータは通常の版管理システムと障害管理システムの情報のみである．そのため，本稿の実験の再現性は高い．今後の課題としては，さらなる開発履歴メトリクスの収集や他の予測モデルの構築，様々なドメインのプロジェクトへの適用などがあげられる．

謝辞 本稿の初版に対して，有益な助言とコメントをい

ただいた査読者の皆様に感謝する．本研究は，日本学術振興会科学技術研究費補助金特別研究員奨励費（課題番号：23・4335）および科学研究費補助金（基盤研究（B）23300009）の助成を受けて実施された．

参考文献

[1] Arcuri, A. and Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering, *Proc. 33rd Int. Conf. on Softw. Eng.*, *ICSE '11*, pp.1–10 (2011).

[2] Arisholm, E., Briand, L.C. and Johannessen, E.B.: A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, *J. Syst. Softw.*, Vol.83, pp.2–17 (2010).

[3] Bettenburg, N., Premraj, R., Zimmermann, T. and Kim, S.: Duplicate bug reports considered harmful . . . really?, *Proc. 24th IEEE Int. Conf. on Softw. Maintenance*, *ICSM '08*, pp.337–345 (2008).

[4] Conway, M.: How do committees invent, *Datamation magazine*, Vol.14, No.4, pp.28–31 (1968).

[5] Czerwonka, J., Das, R., Nagappan, N., Tarvo, A. and Teterev, A.: CRANE: Failure Prediction, Change Analysis and Test Prioritization in Practice – Experiences from Windows, *Proc. 4th IEEE Int. Conf. on Softw. Testing, Verification and Validation*, *ICST '11*, pp.357–366 (2011).

[6] D'Ambros, M., Lanza, M. and Robbes, R.: An extensive comparison of bug prediction approaches, *Proc. 7th IEEE Work. Conf. on Mining Softw. Repositories*, *MSR '10*, pp.31–41 (2010).

[7] Fenton, N., Neil, M., Marsh, W., Hearty, P., Radliński, L. and Krause, P.: On the effectiveness of early life cycle defect prediction with Bayesian Nets, *Empirical Softw. Eng.*, Vol.13, pp.499–537 (2008).

[8] Graves, T.L., Karr, A.F., Marron, J.S. and Siy, H.: Predicting Fault Incidence Using Software Change History, *IEEE Trans. Softw. Eng.*, Vol.26, pp.653–661 (2000).

[9] Hassan, A.E. and Holt, R.C.: The Top Ten List: Dynamic Fault Prediction, *Proc. 21st IEEE Int. Conf. on Softw. Maintenance*, *ICSM '05*, pp.263–272 (2005).

[10] Hata, H., Mizuno, O. and Kikuno, T.: Reconstructing Fine-Grained Versioning Repositories with Git for Method-Level Bug Prediction, *Proc. 2nd Int. Workshop on Empirical Softw. Eng. in Practice*, *IWESEP '10*, pp.27–32 (2010).

[11] Hata, H., Mizuno, O. and Kikuno, T.: Historage: Fine-grained version control system for Java, *Proc. 3rd Joint Int. and Annual ERCIM Workshops on Principles of Softw. Evolution and Softw. Evolution Workshops*, *IWPSE-EVOL '11*, pp.96–100 (2011).

[12] Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K., Adams, B. and Hassan, A.E.: Revisiting common bug prediction findings using effort-aware models, *Proc. 26th IEEE Int. Conf. on Softw. Maintenance*, *ICSM '10*, pp.1–10 (2010).

[13] Kim, S., Zimmermann, T., Pan, K. and Whitehead, E.J.J.: Automatic Identification of Bug-Introducing Changes, *Proc. 21st IEEE/ACM Int. Conf. on Automated Softw. Eng.*, *ASE '06*, pp.81–90 (2006).

[14] Kim, S., Zimmermann, T., Whitehead Jr., E.J. and Zeller, A.: Predicting Faults from Cached History, *Proc. 29th Int. Conf. on Softw. Eng.*, *ICSE '07*, pp.489–498 (2007).

[15] Knab, P., Pinzger, M. and Bernstein, A.: Predicting defect densities in source code files with decision tree learners, *Proc. 3rd Int. Workshop on Mining Softw. Repositories*, *MSR '06*, pp.119–125 (2006).

[16] Liaw, A. and Wiener, M.: Classification and Regression by randomForest, *R news*, Vol.2, No.3, pp.18–22 (2002).

[17] Mende, T. and Koschke, R.: Effort-Aware Defect Prediction Models, *Proc. 14th European Conf. on Softw. Maintenance and Reengineering*, *CSMR '10*, pp.107–116 (2010).

[18] Meneely, A., Williams, L., Snipes, W. and Osborne, J.: Predicting failures with developer networks and social network analysis, *Proc. 16th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, *SIGSOFT '08/FSE-16*, pp.13–23 (2008).

[19] Mockus, A.: Organizational volatility and its effects on software defects, *Proc. 18th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, *FSE '10*, pp.117–126 (2010).

[20] Moser, R., Pedrycz, W. and Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction, *Proc. 30th Int. Conf. on Softw. Eng.*, *ICSE '08*, pp.181–190 (2008).

[21] Nagappan, N. and Ball, T.: Use of relative code churn measures to predict system defect density, *Proc. 27th Int. Conf. on Softw. Eng.*, *ICSE '05*, pp.284–292 (2005).

[22] Nagappan, N., Murphy, B. and Basili, V.: The influence of organizational structure on software quality: An empirical case study, *Proc. 30th Int. Conf. on Softw. Eng.*, *ICSE '08*, pp.521–530 (2008).

[23] Ostrand, T.J., Weyuker, E.J. and Bell, R.M.: Predicting the Location and Number of Faults in Large Software Systems, *IEEE Trans. Softw. Eng.*, Vol.31, pp.340–355 (2005).

[24] Pinzger, M., Nagappan, N. and Murphy, B.: Can developer-module networks predict failures?, *Proc. 16th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, *SIGSOFT '08/FSE-16*, pp.2–12 (2008).

[25] Rahman, F., Posnett, D., Hindle, A., Barr, E. and Devanbu, P.: BugCache for inspections: Hit or miss?, *Proc. 8th Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Found. of Softw. Eng.*, *ESEC/FSE '11*, pp.322–331 (2011).

[26] Ruthruff, J.R., Penix, J., Morgenthaler, J.D., Elbaum, S. and Rothermel, G.: Predicting accurate and actionable static analysis warnings: An experimental approach, *Proc. 30th Int. Conf. on Softw. Eng.*, *ICSE '08*, pp.341–350 (2008).

[27] Śliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes?, *Proc. 2nd Int. Workshop on Mining Softw. Repositories*, *MSR '05*, pp.1–5 (2005).

[28] The R Project for Statistical Computing: R, available from ⟨http://www.r-project.org/⟩.

[29] Weyuker, E.J., Ostrand, T.J. and Bell, R.M.: Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models, *Empirical Softw. Eng.*, Vol.13, pp.539–559 (2008).

[30] Wolf, T., Schroter, A., Damian, D. and Nguyen, T.: Predicting build failures using social network analysis on developer communication, *Proc. 31st Int. Conf. on Softw. Eng.*, *ICSE '09*, pp.1–11 (2009).

[31] Wu, R., Zhang, H., Kim, S. and Cheung, S.-C.: ReLink: Recovering links between bugs and changes, *Proc. 8th Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Found. of Softw. Eng.*, *ESEC/FSE '11*, pp.15–25 (2011).

[32] Zimmermann, T., Nagappan, N., Gall, H., Giger, E. and Murphy, B.: Cross-project defect prediction: A large scale experiment on data vs. domain vs. process, *Proc. 7th Joint Meeting of the European Softw. Eng. Conf. and the ACM SIGSOFT Symp. on the Found. of Softw. Eng.*, *ESEC/FSE '09*, pp.91–100 (2009).
[33] 畑 秀明，水野 修，菊野 亨：リポジトリ再構築によるメソッドトレーサビリティの実現，ソフトウェアエンジニアリングシンポジウム 2010（SES2010），東洋大学，東京，情報処理学会，pp.57–62 (2010).
[34] 小林隆志，林 晋平：データマイニング技術を応用したソフトウェア構築・保守支援の研究動向，コンピュータソフトウェア，Vol.27, No.3, pp.13–23 (2010).

畑 秀明 （学生会員）

平成 19 年大阪大学工学部電子情報エネルギー工学科卒業．平成 21 年同大学大学院博士前期課程修了．現在，同大学院博士後期課程に在学．ソフトウェアの不具合予測手法，ソフトウェアリポジトリのマイニングに関する研究に従事．電子情報通信学会，ACM，IEEE 各会員．

水野 修 （正会員）

平成 10 年大阪大学大学院情報数理系専攻博士前期課程修了．平成 11 年 3 月同大学院博士後期課程中退．博士（工学）．同年 4 月より大阪大学大学院基礎工学研究科助手．その後，大阪大学大学院情報科学研究科助教を経て．平成 21 年 9 月京都工芸繊維大学准教授．主にソフトウェア開発プロセスの改善支援，ソフトウェアの不具合予測手法，ソフトウェアリポジトリのマイニングに関する研究に従事．電子情報通信学会，IEEE 各会員．

菊野 亨 （フェロー）

昭和 50 年大阪大学大学院博士課程修了．工学博士．同年広島大学工学部講師．同大学助教授を経て，昭和 62 年大阪大学基礎工学部情報工学科助教授．平成 2 年同大学教授．現在，大阪大学大学院情報科学研究科教授．大阪大学国際交流センター・センター長．主にフォールトトレラントシステム，ソフトウェア開発プロセスの定量的評価に関する研究に従事．電子情報通信学会，情報処理学会各フェロー．ACM，IEEE 各会員．日本信頼性学会前会長．