開発履歴メトリクスに基づく Fault-prone モジュール予測の細粒度モジュールへの適用

畑　秀明[†1]　水野　修[†2]　菊野　亨[†1]

Fault-prone モジュールの予測モデルにおいて，ソフトウェアリポジトリから収集可能な開発履歴に関する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，コードの変更履歴に関するもの，変更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．一方，版管理システムはソースコードをファイルレベルで管理するため，ファイルレベルの開発履歴メトリクスは収集が容易であるのに対して，関数レベルの細粒度なモジュールにおける開発履歴メトリクスの収集は困難であった．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て，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，細粒度モジュールレベルでの 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．

Historical Metrics Based Foult-prone Module Prediction for Fine-grained Modules

Hideaki Hata,[†1] Osamu Mizuno[†2] and Tohru Kikuno[†1]

Many studies reported that historical metrics collected from software repositories are useful for fault-prone prediction models. 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 not been easy to collect historical metrics of fine-grained modules compared to file-level historical metrics. This paper conducts a comparative study of fault-prone module prediction on files and file-grained modules based on our fine-grained version control system. 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 models on files.

1. はじめに

版管理システムやバグ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[32)]．

Fault-prone モジュール予測の研究においても，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，多くの研究で予測モデル構築に用いられている．文献 4) 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て実際に Windows Vista の開発とメイ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，変更と組み合わせてテストの優先付けを行う．メイ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．

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．これらの履歴の情報はファイルレベルで蓄積されるため，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．

近年，工数を考慮した予測結果の評価法が提案されている[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よい結果を得たことを報告している[10)]．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 特定への工数は大きくなるだろうという直感にも合う．

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†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

このように工数を考慮すると，関数などの細粒度なレベルでの 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[31] 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．我々は同様の試みを文献 9) で行っている．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，メソッドに対する，変更行数に関するメトリクス，不具合修正プロセスに関するメトリクス，開発者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，ファイルレベルと細粒度モジュール（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よい結果が得られた．

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

2. 開発履歴メトリクス

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を計測対象ごとに分類する．

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不具合予測を行っている[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[5),10),11),14),18),20),21),25),30)]．

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 らの成果が挙げられる．Graves らは計測メトリクスとして，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予測モデルを構築している[7]．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不具合予測により有効であることを報告している．

変更回数[5),7),8),10),13),18),20),22)–24),28)]，過去の Fault 数[6),7),14),23),30)]，モジュールの存在期間[5),7),8),10),13),23),25),28)]，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，不具合修正の変更回数[5),8),10),13),15),20),28)]，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）[13),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[7]．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[3] 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，オーナーシップの組織階層，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),29)]．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[29)] を行っている．

2.4 他のメトリクス

地理的に分散した開発と不具合への調査も行われている[2),19)]．Bird らは，ビル，カフェテリア（ビルが異なっても一緒に食事が可能な範囲），キャンパス，都市，大陸，世界を開発の地理的位置として分類し，位置の違いとリリース後の障害との相関を調査している[2)]．Windows Vista でのケーススタディではこれらに相関が見られないことを報告している．一方，電話システムでの調査から，Mockus は分散開発がソフトウェアの品質に影響すると報告している[19)]．

2.5 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[10),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からのメトリク

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
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）
	AddLOC	最初の版からの追加行数
	DelLOC	最初の版からの削除行数
	AddPerLOC	AddLOC / LOC
	DelPerLOC	DelLOC / LOC
プロセス関連	ComNum	変更回数
	FixComNum	Fault 修正の変更回数
	FaultNum	過去の Fault 数（関連するバグレポート数）
	Period	存在期間（週単位）
	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
	BugIntroTiming	Fault が混入された変更時に変更された回数
開発組織関連	AuthorNum	そのモジュールを変更した開発者数

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解析対象とする．筆者らは，既存のファイルレベルの版管理システムの情報から，細粒度モジュールレベ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[31]．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，細粒度モジュールにおいても計測可能となる．

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．コードの変更行数に関するものと変更プロセスに関する主なメトリクスを計測する．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[26] で特定する．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はないため，開発組織関連のメトリクスとしては，開発者数のみを計測する．

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[26] が広く使われている．版管理システムとバグ管理システムの情報を用いて，バグ管理システムに登録されたバグの原因となった Fault の情報を取得する．おおまかには，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**　登録されたバグに関する Fault 修正の変更を探す．版管理システムのコミットログにバグレポートの 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，対応するバグレポートが最初に報告された日付より後であり，またそのバグ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，そのバグレポートに対する 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，対応するバグ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，バグ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，そのバグに関する 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 12) 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[27] 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 10)，17) などでも同パッケージが用いられている．

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

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

			ファイル		メソッド	
プロジェクト	タグ	作成日	Fault あり	Fault 無し	Fault あり	Fault 無し
xpd	Galileo_RC1	2009-05-18	115	1,132	295	8248
wti	v20090510	2009-05-10	140	466	317	5175
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

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文献 10) の方法を参考にする．Arisholm らは，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ことを指摘しており[1)]，文献 10) でもモジュールの行数（LOC）を工数と定めている．

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 を特定したい．そこで，調査モジュールの累積行数が全体行数に対して一定の割合のとき，全体の内どれだけの Fault ありモジュールを特定できるか（Recall）で評価する．文献 10) 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 Recall の値を調べる．

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 Communication 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．またバグ管理システム Bugzilla[⋆5]から 2011 年 4 月 30 日までのバグ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．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．3.2 節に示したアルゴリズムで取得した，Fault 有無のモジュール数をそれぞれファイルレベルとメソッドレベルで示す．

⋆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 4　累積行数 20% での Recall 値（%）
Table 4　Recall value (%) on 20% cumulative LOC.

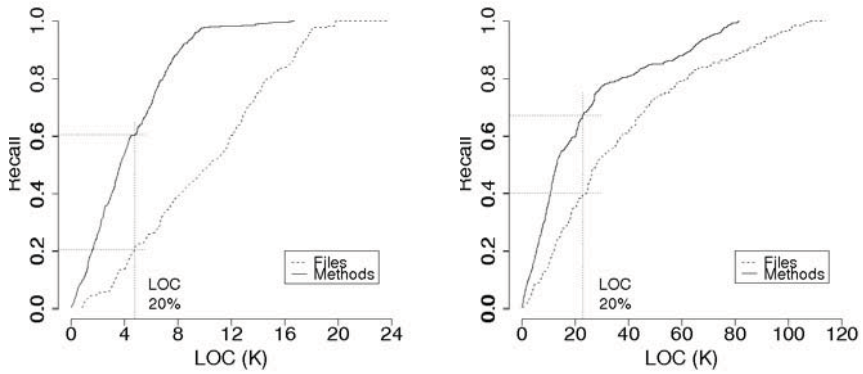
プロジェクト	ファイルレベル	メソッドレベル	差分（メソッドレベル - ファイルレベル）
xpd	15.7	51.9	36.2
wti	40.0	67.5	27.5
emfc	20.3	60.4	40.1
ecf	40.0	67.0	27.0

4. 結　　果

Fault-prone モジュールを確率が大きい順に並べ，累積行数が 20% を超えたときの Recall 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．全てのプロジェクトで，メソッドレベルがファイルレベルに比べて高い Recall 値を得た．すなわち，同程度の工数でメソッドレベルの方がより Fault を特定できるといえる．メソッドレベルは，ファイルレベルから Recall 値が 27% から 40% ほど向上しており，累積行数 20% までに Fault ありモジュールの 50% 以上が含まれていることが分かる．

図 1 に，`emfc` プロジェクトと `ecf` プロジェクトでの累積行数と Recall の推移をまとめたグラフを示す．累積行数が 20% となるときの Recall を点線で示す．図 1 から，メソッドレベルはファイルレベルと比べて，同一累積行数で高い Recall となり，また少ない累積行数で同一 Recall 値を得ることができるということが確認できる．他のプロジェクトでも同

⋆4 http://git.eclipse.org/c/ecf/org.eclipse.ecf.git/
⋆5 https://bugs.eclipse.org/bugs/



(a) emfc プロジェクト　　(b) ecf プロジェクト

図 1　累積行数と Recall の推移
Fig. 1　LOC-based cumulative lift chart for file-level v.s. method-level.

様の結果が得られた．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 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の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た．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．最上位のメトリクスは 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．

今回の評価実験は，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．全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．

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. 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細粒度モジュールレベルでの 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．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今後の課題である．

謝辞　本論文の初版に対して，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）の助成を受けている．

参 考 文 献

1) 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).
2) Bird, C., Nagappan, N., Devanbu, P., Gall, H. and Murphy, B.: Does distributed development affect software quality? An empirical case study of Windows Vista, ICSE '09, pp.518–528 (2009).
3) Conway, M.: How do committees invent, *Datamation*, Vol. 14, No. 4, pp. 28–31 (1968).
4) Czerwonka, J., Das, R., Nagappan, N., Tarvo, A. and Teterev, A.: CRANE: Failure Prediction, Change Analysis and Test Prioritization in Practice – Experiences from Windows, ICST '11, pp.357–366 (2011).
5) D'Ambros, M., Lanza, M. and Robbes, R.: An extensive comparison of bug prediction approaches, MSR '10, pp.31 –41 (2010).
6) 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).
7) 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).
8) Hassan, A.E. and Holt, R.C.: The Top Ten List: Dynamic Fault Prediction, ICSM '05, pp.263–272 (2005).
9) Hata, H., Mizuno, O. and Kikuno, T.: Reconstructing Fine-Grained Versioning Repositories with Git for Method-Level Bug Prediction, IWESEP '10, pp. 27–32 (2010).
10) Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.-i., Adams, B. and Hassan, A.E.: Revisiting common bug prediction findings using effort-aware models, ICSM '10, pp.1–10 (2010).
11) Kim, S., Whitehead, Jr., E.J. and Zhang, Y.: Classifying Software Changes: Clean or Buggy?, *IEEE Trans. Softw. Eng.*, Vol.34, pp.181–196 (2008).
12) Kim, S., Zimmermann, T., Pan, K. and Whitehead, E. J.J.: Automatic Identification of Bug-Introducing Changes, ASE '10, pp.81–90 (2006).
13) Kim, S., Zimmermann, T., WhiteheadJr., E.J. and Zeller, A.: Predicting Faults from Cached History, ICSE '07, pp.489–498 (2007).
14) Kläs, M., Elberzhager, F., Münch, J., Hartjes, K. and von Graevemeyer, O.: Transparent combination of expert and measurement data for defect prediction: an industrial case study, ICSE '10, pp.119–128 (2010).
15) Knab, P., Pinzger, M. and Bernstein, A.: Predicting defect densities in source code files with decision tree learners, 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, 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, SIGSOFT '08/FSE-16, pp.13–23 (2008).
19) Mockus, A.: Organizational volatility and its effects on software defects, 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, ICSE '08, pp. 181–190 (2008).
21) Nagappan, N. and Ball, T.: Use of relative code churn measures to predict system defect density, 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, 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?, SIGSOFT '08/FSE-16, pp.2–12 (2008).
25) Ruthruff, J.R., Penix, J., Morgenthaler, J.D., Elbaum, S. and Rothermel, G.: Predicting accurate and actionable static analysis warnings: an experimental approach, ICSE '08, pp.341–350 (2008).
26) Śliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes?, MSR '05, pp.1–5 (2005).
27) The R Project for Statistical Computing: R. `http://www.r-project.org/`.
28) 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).
29) Wolf, T., Schroter, A., Damian, D. and Nguyen, T.: Predicting build failures using social network analysis on developer communication, ICSE '09, pp.1–11 (2009).
30) 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, ESEC/FSE '09, pp.91–100 (2009).
31) 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)，情報処理学会，pp.57–62 (2010).
32) 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, pp.13–23 (2010).