

論文

11 種類の fault 密度予測モデルの実証的評価

小林 寛武[†] 戸田 航史^{††} 亀井 靖高^{†a)} 門田 暁人^{†††}
 峯 恒憲[†] 鵜林 尚靖[†]

Experimental Evaluation of Eleven Fault-Density Models

Hirotake KOBAYASHI[†], Koji TODA^{††}, Yasutaka KAMEI^{†a)}, Akito MONDEN^{†††},
 Tsunenori MINE[†], and Naoyasu UDAYASHI[†]

あらまし ソフトウェアテスト・保守において、限られたリソースで信頼性を確保することを目的として、モジュールに含まれる fault の有無や数を予測するモデルが数多く提案されている。しかし、従来研究では、評価用のプロジェクト（データセット）の数が少ない、比較に用いたモデル数が少ない、モジュールの規模を考慮した評価を行っていない（fault 密度を予測していない）、モデル構築用と評価用のデータセットを同じサンプルから抽出している（バージョンをまたがる評価を行っていない）、といった問題があり、いずれの fault 予測モデルが優れているのか明らかでなかった。そこで本論文では、25 のオープンソースプロジェクトからそれぞれ 2 バージョンずつ収集したデータセットを用いて、11 種類の fault 密度予測モデルの性能を実験的に評価した。実験の結果、樹木モデル（M5）が最も高い性能を有することが分かった。

キーワード fault 密度予測, 実証的実験, オープンソースプロジェクト, バージョン間の予測, Demšar's significance diagram

1. ま え が き

ソフトウェアテスト及び保守工程において、fault-prone モジュール（欠陥を含む確率の高いモジュール）の予測は、テスト工数の割り当てやレビューの優先順位の決定に役立つ [1]。そのため、モジュールのメトリックス（ソースコード行数、サイクロマティック数など）を説明変数とし、モジュール中の fault 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くの論文で提案されており、企業における適用事例も報告されている [2], [3]。また、fault の有無だけでなく、fault 数 [4], [5] や fault 密度 [6] の予測モデルも

提案されてきた。しかしながら、従来研究では、予測モデル評価のための実験環境に問題があり、いずれの予測モデルが最も優れているかが必ずしも明らかでなかった。具体的には、次の 3 点が問題である。

（1）性能比較を行っている予測モデルの数が少ない、若しくは、性能評価に用いるデータセットの数が少ない。結果として、最も性能が良いとされる予測モデルが、論文ごとに異なっている [7]。

（2）従来研究の多くはモジュールの fault の有無を予測しているが、予測後に必要となるテスト工数を考慮した評価が行えない点が問題である。少ないテスト工数でより多くのバグを発見するという観点からは、規模が小さくてバグが多いモジュール、すなわち fault 密度の高いモジュールを予測する方が開発現場にとって役立つ [6]。しかし、fault 密度予測モデルの性能評価やテスト工数を考慮したモデルの評価は従来あまり行われていない。

（3）従来研究では、予測モデルの評価実験において、交差検証（cross-validation）がよく用いられてきた。しかし、交差検証ではモデル構築用と評価用のデータセットを同じサンプルから抽出するため、開発現場の実態にそぐわない点が問題である。開発現場で

[†] 九州大学大学院システム情報科学研究院, 福岡市
 Graduate School and Faculty of Information Science and
 Electrical Engineering, Kyushu University, 744 Motooka,
 Nishi-ku, Fukuoka-shi, 819-0395 Japan

^{††} 福岡工業大学情報科学研究科, 福岡市
 Department of Computer Science and Engineering, Fukuoka
 Institute of Technology, 3-30-1 Wajiro-Higashi, Higashi-ku,
 Fukuoka-shi, 811-0295 Japan

^{†††} 奈良先端科学技術大学院大学情報科学研究科, 生駒市
 Graduate School of Information Science, Nara Institute of
 Science and Technology, 8916-5 Takayama, Ikoma-shi, 630-
 0192 Japan

a) E-mail: kamei@ait.kyushu-u.ac.jp

は、過去に開発された（旧バージョンの）ソフトウェアのデータを用いてモデル構築し、開発中の（現バージョンの）ソフトウェアの fault を予測することが想定されるため、このようなバージョンをまたがった予測モデルの評価が必要である。

これら現状の課題を踏まえ、本論文では 25 のオープンソースプロジェクトから計測したデータセットを用いて、次の二つの Research Question (RQ) に取り組む。

RQ1: テスト工数を考慮した fault 密度予測モデルの評価

RQ1 では、11 種類の fault 密度予測モデルの予測性能の優劣を実験的に明らかにする。実験では、予測後に必要となるテスト工数を考慮した評価尺度を用いる。

RQ2: バージョンをまたがる fault 密度予測モデルの評価

RQ2 では、実プロジェクトでの利用を考慮し、過去に開発された（旧バージョンの）ソフトウェアのデータを用いてモデル構築し、次期バージョンにおける各モジュールの fault 密度を予測し、交差検証による予測結果との違いを明らかにする。

2. 関連研究

2.1 fault-prone 予測モデル

従来、fault-prone モジュール予測モデルの構築方法が数多く提案されてきた [8]~[13]。例えば、ロジスティック回帰分析は多くの研究で利用されている手法の一つで、与えられた説明変数の値の組（モジュールから計測されるメトリックス値の組）に対する、ある現象の発生する条件付き確率（モジュールが fault を含む確率）をロジスティック回帰式としてモデル化する手法である [8], [9], [11]。それ以外にも、線形判別分析 [12]、分類木 [10]、ニューラルネットワーク [14]、Support Vector Machine [15]、ベイズ識別器 [16], [17]、 k 近傍識別器 [18] などのモデル化手法が用いられており、最近の研究では判別モデルにおけるランダムフォレストの優位性が示唆されている [19]。

本論文では、fault 密度を予測するため、2 値判別（fault の有無）に用いるモデルではなく、数値の予測を行える 11 種類のモデルを採用した。

2.2 テスト工数を考慮したモデルの評価

規模が大きいモジュールほどより多くのテスト工数が必要となることから、モジュールの規模を考慮して fault-prone モジュール予測モデルを評価する事例と

して、Mende ら [20] による研究がある。Mende らはモジュールのソースコード行数がテスト工数に比例するとみなした上で、モデルの予測精度の評価を行っている。

また、Kamei ら [21] は、Mende らの方法を拡張して fault-prone モジュール予測を行い、いくつかの知見を得ている。例えば、プロセスメトリックスはテスト工数を考慮した場合でも、プロダクトメトリックスよりも性能の良い予測モデルを構築できることを示している。

本論文では、実験に用いるプロジェクト数を 25 件に増やした上で、テスト工数（モジュールの規模）を考慮して予測モデルの性能比較を行う。また、本論文では、予測モデルの説明変数として、7 種類のプロセスメトリックスを採用する。

2.3 バージョンをまたがるモデルの評価

従来、fault-prone モジュール予測モデルの評価方法として交差検証法（N-fold cross-validation）が広く用いられてきた [19], [20], [22]。一方で、複数のバージョンを用いて予測モデルの性能を評価する方法も近年行われている [21], [23], [24]。例えば Kamei ら [21] は 3 種類のオープンソースのソフトウェアを対象に、特定のあるバージョンでのモジュールの判別には、それより古いバージョンのデータから構築したモデルを用いる、という方法を採用している。

本論文においても、より現実に即した評価を行うために、過去バージョンのデータを用いて予測モデルを構築し、次期バージョンに適用した場合の予測性能を評価する。

3. Research Question

本論文では、実プロジェクト環境下において有用な予測モデルの特定を目的として、二つの Research Question (RQ) に取り組む。本章ではその二つの RQ の概要・動機について説明する。

3.1 RQ1: テスト工数を考慮した fault 密度予測モデルの評価

従来行われてきた fault-prone モジュール判別研究の多くは、各モジュール中の fault の有無という 2 値判別モデルの性能を評価している。しかしながら、一般に規模が大きいモジュールほど多くの fault を含む傾向にあり [28]、fault-prone モジュール判別では規模の大きいモジュールばかりが fault-prone と判断される可能性が高い。この場合、モジュールのテストに必

表 1 実験で用いた fault 密度予測モデルとそのハイパパラメータ
Table 1 Prediction methods and hyperparameter in our experiments.

カテゴリー	比較手法	概要	ハイパパラメータ
統計的アプローチ	LM	線形回帰	なし
	LARS	Least Angle Regression [25]	なし
	RVM	Relevance vector machine [26]	なし
近傍法	k-NN	k 近傍法	$k = \{1, \dots, 15\}$
	K*	エントロピーに基づく距離を使用した k 近傍法 [27]	なし
ニューラルネット	NNET	3 層ニューラルネット	$size = \{4, \dots, 28\}, decay = \{0.1, 0.2\}$
サポートベクターマシン	SVM	サポートベクターマシン	$C = \{2^{-6}, \dots, 2^{10}\}$
樹木モデル	M5	分類木	なし
	RPART	CART	$cp = \{0.05, 0.1, \dots, 0.7\}$
集団学習	GBM	ブースティングを用いた分類木	$n.tree = \{10, \dots, 50\}, shrinkage = \{0.1\}$
	RF	ランダムフォレスト	$mtry = \{10, 50, 100, 200, 250, 500, 1000\}$

要な工数はそのモジュールのソースコード行数に比例する [29] ため、開発者は判別結果に基づいて効率的なテスト計画を作成することが困難な場合がある。この問題を解決するために、Kamei らはテスト工数を考慮した fault モジュール予測モデルの評価を行っている [21]。Kamei らの実験ではソースコード行数をテスト工数とみなし、fault 密度 (fault 数/ソースコード行数) 予測を行っている。開発者は fault 密度の予測性能が高いモデル構築手法を用いることで、単位行数当りに含まれる fault 数が大きいモジュールから優先的にテスト工数を割り振ることが可能となり、効率的なテスト計画を作成することができる。しかし、多数のデータセット、多数の fault 密度予測モデルを用いた場合での予測性能を比較する研究は従来十分には行われてこなかった。そこで、RQ1 ではソースコード行数を考慮した 11 種類の fault 密度予測モデルを比較し、予測性能の優劣を実験的に明らかにする。用いる各予測モデル構築手法の名称とその概要、ハイパパラメータ (4.2.2 にて詳述) の推定候補値を表 1 に示す。11 の手法は予測モデルの六つのカテゴリー (統計的アプローチ, 近傍法, ニューラルネット, Support Vector Machine, 樹木モデル, 集団学習) を網羅するように選んだ。

3.2 RQ2: バージョンをまたがる fault 密度予測モデルの評価

開発者が実案件で fault 密度予測モデルを用いる場合、過去に開発された (過去バージョンの) ソフトウェアのデータを用いてモデルを構築する。しかし、従来研究では同一バージョンでの交差検証法による評価が広く行われている [19]。そこで RQ2 では、過去バージョンのデータ (メトリックス) を利用して fault 密

度予測モデルを構築し、次期バージョンに適用した場合の予測性能を評価する。

更に、RQ1 で検証した結果との差異や相関について調査することにより、開発者が過去のバージョンの情報をもとに、fault 密度予測モデルを選択する場合の方針を明らかにする。

4. 実験

実験では、25 の OSS プロジェクトで収集されたデータに対して、11 種類の手法で fault 密度予測し、その精度を比較した。本章では RQ を検証するために用いたデータセットや精度向上のために行ったモデル構築前の処理、評価基準、実験の実行環境について説明する。

4.1 データセット

4.1.1 対象プロジェクト

25 の OSS プロジェクトそれぞれから収集された各メトリックス値を記録したものを実験に用いるデータセットのベースとした。各プロジェクトのモジュール数と fault 数を表 2 に示す。また、25 のプロジェクトのドメインは限定しなかった。これは本研究のアプローチとして、広いドメインすなわち状況に限定されない環境下での精度評価を目的とするためである。ただし、今後同様の分析を行う場合の利便性を考え、ドメイン以外の共通点として Git 若しくは CVS によるバージョン管理を行い、かつ Bugzilla や JIRA 等によるバグ管理を行っているプロジェクトに限定した。各プロジェクトの対象バージョンやメトリックス収集期間など更に詳細な情報は付録にて説明する。全てのデータセットについて、B は A よりも新しいバージョンである。

論文/11 種類の fault 密度予測モデルの実証的評価

表 2 実験対象のプロジェクトの各バージョンでのファイル数とモジュール数

Table 2 The number of files and modules of two versions' projects used in experiments.

プロジェクト名	略称	バージョン A		バージョン B	
		module	fault	file	module
Clam AntiVirus	CLAM	1,597	93	8,723	56
eCos	ECOS	630	110	3,480	67
Exim Internet Mailer	EXIM	184	28	61	31
freedesktop	FREE	1,982	477	2,855	603
Ganymede	GANY	184	29	95	30
Helma	HELM	312	38	100	17
Hibernate Core	HIBE	406	87	5,032	1,191
HylaFAX	HYLA	137	70	109	35
iptables	IPTA	151	96	276	91
KOffice	KOFF	930	363	990	232
NetBSD	NBSD	6,781	548	10,960	299
OpenBSD	OBSD	1,706	275	5,964	158
OpenCms	OCMS	1,727	193	2,821	93
openNMS	ONMS	1,203	123	1,416	112
OTRS	OTRS	420	111	763	169
PostgreSQL	POST	863	435	1,211	325
ProFTPD	PROF	195	64	220	61
Samba	SAMB	1,623	543	2,559	439
Scilab	SCIL	2,636	239	1,248	244
Spring Security	SPRI	926	126	639	132
squid	SQUI	250	59	185	92
Wine	WINE	1,627	584	1,382	288
XDoclet	XDOC	105	30	135	5
XFree86	XFRE	713	350	761	259
XORP	XORP	1,696	158	1,755	217

4.1.2 計測メトリックス

本論文では 1 ソースコードファイルを 1 モジュールとみなし、Moser らの提案するプロセスメトリックス [22] のうち代表的な 7 種類をモジュール単位のプロセスメトリックスとして収集した。収集の対象としてはバージョン管理ツール (CVS, Git など) のリポジトリにおけるコミットログを用いた。各メトリックスの名称と概要を表 3 に示す。これらのメトリックスは、プロジェクトがバージョン管理システムを導入していれば、コミットログだけから算出でき、計測が容易である。一方、サイクロマティック数やメソッド数などのプロダクトメトリックスは、計測にコストを要することと、開発中の (不完全な) ソースコードからは必ずしも計測できないこと、バグの混入に対してプロセスメトリックスほど寄与しない [30]、プログラミング言語ごとに専用の計測ツールが必要、などの理由から本論文では計測の対象とはしなかった。

なお、表 3 の BugFixes, Refactorings の値には、従来研究 [22], [31], [32] と同様にモジュール単位で計測したコミットログ中の特定の語 (“Fix” や “Refac-

表 3 用いたプロセスメトリックス

Table 3 Measured metrics.

名称	概要
Codechurn	変更行数 (追加行数+削除行数)
LOCAdded	追加行数
LOCDeleted	削除行数
Revisions	改訂回数
Age	最後に修正されてからの経過日数
BugFixes	fault 修正回数
Refactorings	リファクタリング回数

toring”) を含む変更を fault 修正, 若しくはリファクタリングとみなした。このうち BugFixes については、OSS プロジェクトでは fault 修正のコミット時には、fault を修正した旨と fault の ID をコミットコメントに残すことがルールづけられており、キーワードマッチによりある程度網羅的に検出できていると考えられる。ただし、ルールに従っていない fault 修正については、見逃してしまうことになる。また、Refactoring については、ルール付けされているとは限らないため、見逃しも少なくない点に注意が必要である^(注1)。また BugFixes と fault の収集方法についても従来研究 [33] と同様に、BugFixes で用いたキーワードに一致するものを、リリース前 (メトリックス収集期間) では BugFixes, リリース後 (fault 収集期間) では fault としてカウントした。また収集期間の詳細については付録の表 A・1, 表 A・2 に記載した。

4.1.3 実験用データセット

RQ1 では、表 2 に示すバージョン A のデータセットからランダムに選んだ 2/3 のモジュールを学習データとして用い、残りの 1/3 をテストデータとして利用した。

RQ2 では、表 2 に示すバージョン A のデータセットを学習データ、バージョン B のデータセットをテストデータとして、モデル構築、予測を行った。

4.2 モデル構築前処理

モデル構築とその精度に関わる、データのの前処理とモデルのハイパパラメータの設定について説明する。

4.2.1 データの前処理

いくつかのプロジェクトでは収集時点までにリファクタリングが行われていなかったため、Refactorings の値が 0 となっていた。値が 0 のメトリックスの採用はモデルの精度に悪影響を及ぼすため、あらかじめ除外した。

(注1): キーワードマッチを用いたプロセスメトリックス計測手法の制約の詳細については文献 [17] を参照のこと。

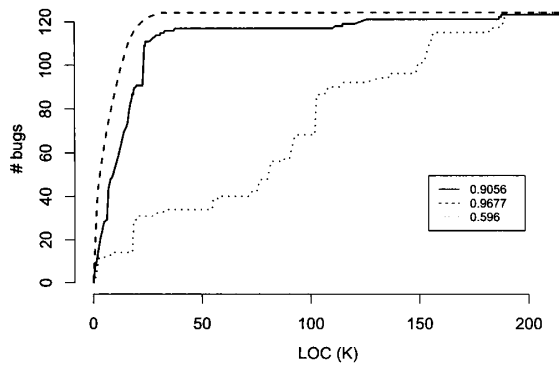


図 1 LOC 曲線の例
Fig.1 Example of LOC curve.

また、メトリックス間の相関が高い場合、一部のモデル構築手法では多重共線性によってモデルの信頼性が低下する。ここでは相関係数 0.8 以上のメトリックスの組合せは多重共線性の原因となると判断し、一方をモデル構築から除外した。

4.2.2 ハイパパラメータの推定

一部の予測モデルでは、モデル構築時にパラメータを設定する必要がある。このうちモデル全体に影響を与えるパラメータはハイパパラメータと呼ばれ、識別精度に大きな影響を与える。そこで、本実験では 10-fold の交差検証法を用いて最適なハイパパラメータ値の推定を行った。各手法のハイパパラメータとその推定の候補値は表 1 に示したとおりである。ただし統計的アプローチにはハイパパラメータが存在しないので「なし」と表記されている。

4.3 実験の評価

実験の評価では、従来研究 [20], [34] と同様に、累積行数が全体行数の 20% の時点で検出できる fault の Recall 値を用いた。累積行数と fault 数の関係は LOC ベースの曲線で表現できる。LOC ベースの曲線は、fault 密度の予測値が大きい順にモジュールを並べ、横軸にモジュールの累積行数、縦軸に累積実測 fault 数を取ったときの変化を表す。LOC ベースの曲線の例を図 1 に示す。図中の破線は予測の理論値 (fault 密度の実測値が大きい順に並べた結果)、実線は予測精度、点線はランダムに評価した場合の精度の例である。

LOC ベースの曲線は従来研究で評価に用いられる Alberg Diagram [12] に類似しているが、Alberg Diagram とは横軸にモジュールの累積行数をとる点が異なる。Alberg Diagram と比較した場合、LOC ベースの曲線では規模は大きい fault 数がそれほど多くないモジュール (fault 密度が小さいモジュール) を誤

判別する予測モデルを精度が低いと判定するため、モジュールサイズに依存しない結果が得られる。すなわち、LOC ベースの曲線から算出できる Recall 値は工数に対する fault の発見効率を表す指標となる。

4.4 実験の実行環境

オープンソースの統計処理環境である R [35] を用いて、fault 密度予測モデルの構築及び予測、評価を行った。また、データセットの分割や 10-fold の交差検証法によるハイパパラメータ決定及び予測モデルの作成には、R 上で利用できるパッケージの一つである caret [36] を利用した。

5. 結果と考察

5.1 RQ1: テスト工数を考慮した fault 密度予測モデルの評価

LOC ベースの曲線の累積行数が 20% を超えた時点で発見できる fault の割合 (Recall) を表 4 に示す。表中では 1 行が一つのデータセットに、1 列が一つのモデル構築手法による予測結果 (AUC 値) に対応する。最下行の AR はある手法の各データセットにおける精度での順位平均を表す。すなわち i 番目の手法を AR_i 、全データセットの個数を K とすると AR_i は式 (1) で表現される。ここで r_j^i は i 番目の手法に対して j 番目のデータセットを適用したときの精度の順位を表す。

$$AR_i = \frac{1}{K} \sum_{j=1}^K r_j^i \quad (1)$$

また表中の太字は、AR 以外ではそのデータセットで最も高い精度の値を、AR では最も順位平均が低い (精度が高い) 値を示している。以降の表でも太字は同じ事を意味するものとする。表 4 中の AVG は各予測モデルの Recall の平均値、AR は各データセットでの Recall の平均順位を表す。

ほぼ全てのデータセットにおいて、Recall が 30% を超える予測モデルが存在しており、また、平均で 40% を超える Recall 値を示すモデル構築手法が存在する。この結果は、ランダムに予測した場合と比較して約 2 倍の精度で予測できることを示している。

更に、各予測モデルの比較のために、Lessmann らの提案する比較実験フレームワーク [19] を用いた。具体的には、Nemenyi's test の結果を Demšar's significance diagram を用いて表現し (図 2)、モデル間の優劣を統計的に比較する。Demšar's significance diagram では縦軸を手法、横軸を AR (精度の平均順位)

表 4 RQ1: fault 密度予測における各手法の性能 (%)
Table 4 RQ1: Prediction performance of each model when effort considered (%).

	LM	LARS	RVM	k-NN	K*	NNET	SVM	M5	RPART	GBM	RF
CLAM	89.5	91.1	32.3	34.7	90.3	16.9	66.9	90.3	67.7	70.2	94.4
ECOS	32.5	30.0	17.5	32.5	45.0	25.0	12.5	27.5	32.5	37.5	35.0
EXIM	25.0	25.0	0.0	12.5	50.0	37.5	50.0	0.0	0.0	25.0	25.0
FREE	43.1	48.8	24.7	54.1	54.4	34.1	41.3	56.3	8.4	52.8	54.4
GANY	12.5	12.5	8.3	25.0	16.7	25.0	25.0	25.0	0.0	29.2	33.3
HELM	21.7	21.7	8.7	30.4	26.1	13.0	13.0	21.7	0.0	30.4	21.7
HIBE	41.8	45.5	52.7	16.4	38.2	45.5	34.5	41.8	18.2	49.1	45.5
HYLA	29.9	63.5	17.5	21.2	52.6	21.9	15.3	35.8	35.0	24.1	65.7
IPTA	9.5	19.0	31.0	26.2	16.7	23.8	23.8	11.9	21.4	21.4	21.4
KOFF	23.1	28.3	25.9	32.1	32.1	30.2	23.1	34.0	34.0	28.8	35.4
NBSD	47.7	52.7	19.2	41.0	60.7	26.4	38.5	52.7	10.9	46.4	54.0
OBSD	53.9	60.7	13.5	44.4	59.6	42.1	37.6	63.5	6.2	48.9	60.1
OCMS	27.0	15.7	20.0	16.5	14.8	18.3	15.7	22.6	13.9	16.5	12.2
ONMS	26.9	19.2	26.9	30.8	53.8	46.2	17.3	40.4	15.4	46.2	36.5
OTRS	22.4	19.0	25.9	29.3	22.4	22.4	24.1	20.7	32.8	32.8	17.2
POST	16.1	42.2	25.8	40.1	40.7	41.0	35.9	42.6	15.2	41.3	39.2
PROF	35.9	33.3	9.0	23.1	28.2	6.4	26.9	25.6	25.6	14.1	23.1
SAMB	22.8	24.8	39.6	39.6	34.7	38.6	31.7	38.6	25.7	30.7	21.8
SCIL	39.0	42.9	35.2	41.0	51.4	40.0	26.7	58.1	31.4	58.1	56.2
SPRI	41.3	42.5	22.5	33.8	35.0	51.3	31.3	48.8	17.5	47.5	55.0
SQUI	26.2	29.5	9.8	21.3	14.8	16.4	14.8	16.4	6.6	27.9	27.9
WINE	30.6	50.6	22.2	35.7	39.5	24.4	33.0	51.7	16.2	32.2	35.3
XDOC	30.8	61.5	30.8	23.1	53.8	46.2	38.5	46.2	15.4	61.5	53.8
XFRE	22.8	45.7	34.9	41.9	43.3	42.2	40.8	45.3	37.0	41.2	47.4
XORP	58.6	58.6	25.3	54.0	69.0	34.5	31.0	71.3	56.3	56.3	50.6
AVG	33.2	39.4	23.2	32.0	41.7	30.8	30.0	39.5	21.7	38.8	40.9
AR	6.32	4.6	8.08	5.72	4.24	6.12	7.52	3.92	8.44	4.28	4.2

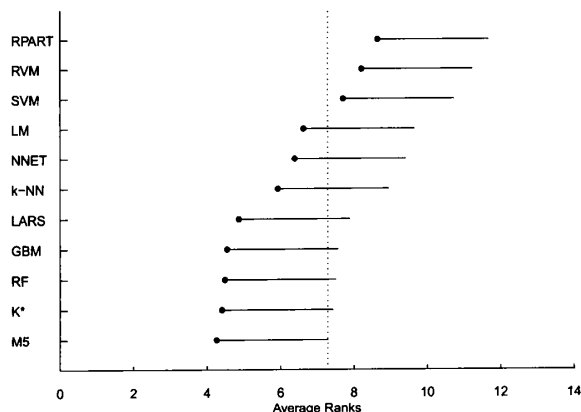


図 2 RQ1 での Demšar's significance diagram
Fig. 2 Demšar's significance diagram in RQ1.

としてグラフを描く。図中では各手法は AR 値でソートされており、図中の ● は各手法の AR 値（平均順位）を意味し、そこから伸びる実線は、各手法について「平均順位（AR 値）が等しい」という帰無仮説を有意水準 5% で棄却できない範囲を示す。図中点線は最も AR 値が低い（精度の平均順位が高い）手法である M5 において、帰無仮説で棄却できない最大の AR 値を表しており、点線より左側に AR 値のある 7 手法

（K*から LM）には統計的に M5 との間に有意な差はない。また、GBM, RF などの集団学習手法や近傍法である K*などが上位に入っている。ただし、統計的アプローチである LM と LARS と RVM, 樹木モデルである M5 と RPART は予測手法のカテゴリーが同一であるにもかかわらず精度に大きな差がみられた。

この原因としては以下のような理由が考えられる。統計的アプローチでは、特に RVM の精度が低くなっている。RVM は説明変数を要素とする特徴ベクトルを非線形変換し、次元を増加させることにより予測性能を向上させる手法であるが、本論文で扱うデータセットは、ノイズとなる値を多く含む（fault の混入は本質的にはヒューマンエラーであるため、大規模で変更が多いモジュールであっても fault が含まれるとは限らず、逆に、小規模だから fault を含まないとも限らない）ため、非線形変換によってモデルの適合度を上げようとするアプローチがうまく働かなかった可能性がある。

次に、樹木モデルでは、RPART が元来判別予測のために開発された 2 分木で構成される手法であるのに対し、M5 は枝の末端に線形回帰式が割り当てられる

表 5 RQ2: 過去バージョン利用時の fault 密度予測における各手法の性能 (%)
 Table 5 RQ2: Prediction performance using the existence of fault as an objective variable for cross-release analysis.

	LM	LARS	RVM	k-NN	K*	NNET	SVM	M5	RPART	GBM	RF
CLAM	76.1	93.6	41.3	43.1	82.6	15.6	66.1	55.0	53.2	97.2	93.6
ECOS	37.5	38.9	18.1	25.0	26.4	11.1	20.8	22.2	18.1	27.8	22.2
EXIM	20.5	20.5	17.9	25.6	7.7	53.8	12.8	10.3	10.3	23.1	15.4
FREE	41.2	50.0	31.8	59.8	58.2	60.2	40.1	60.2	22.7	59.9	59.8
GANY	25.5	17.6	21.6	21.6	13.7	23.5	9.8	19.6	15.7	11.8	17.6
HELM	8.3	8.3	25.0	8.3	12.5	37.5	16.7	8.3	16.7	8.3	4.2
HIBE	16.3	27.5	22.5	24.9	24.4	21.4	19.3	27.6	15.7	20.7	21.9
HYLA	35.0	38.0	27.0	25.0	29.0	28.0	36.0	46.0	34.0	36.0	34.0
IPTA	21.1	21.1	24.0	28.1	26.9	24.0	27.5	25.1	17.0	26.3	24.0
KOFF	25.2	23.9	19.5	28.1	28.1	30.8	23.5	29.5	28.3	22.5	29.3
NBSD	37.5	40.5	24.7	31.5	28.6	23.8	26.5	37.2	9.8	36.6	32.4
OBSD	47.1	51.6	24.0	35.1	40.4	29.3	15.1	51.6	12.9	30.7	39.6
OCMS	57.6	58.9	9.3	39.1	47.0	17.2	56.3	53.6	58.3	31.1	29.1
ONMS	26.8	24.8	22.9	24.2	33.3	21.6	15.7	27.5	20.3	24.8	31.4
OTRS	59.9	63.9	54.3	30.0	55.2	29.8	54.9	60.5	68.2	67.0	35.9
POST	24.8	24.6	30.2	29.6	32.0	32.2	28.2	34.5	23.6	27.9	29.4
PROF	38.1	24.4	20.8	11.9	14.3	22.6	24.4	27.4	31.5	26.8	24.4
SAMB	34.8	40.3	17.1	29.2	33.9	24.7	30.5	31.4	25.8	31.0	34.7
SCIL	26.9	26.9	21.8	24.6	32.1	23.8	28.0	33.2	19.7	33.4	31.6
SPRI	38.0	39.7	17.0	27.5	30.6	32.3	29.7	40.2	27.1	41.9	40.2
SQUI	25.4	38.9	25.2	24.9	26.4	19.7	31.2	35.2	4.5	28.4	21.4
WINE	35.2	41.3	16.2	38.4	33.1	27.2	34.0	45.2	21.7	43.9	37.8
XDOC	50.0	33.3	16.7	0.0	66.7	0.0	0.0	33.3	16.7	33.3	16.7
XFRE	31.7	29.6	30.4	35.0	40.9	32.4	25.6	39.8	33.3	31.5	33.3
XORP	49.2	54.1	16.9	42.9	50.0	36.6	38.3	63.9	48.1	40.2	50.3
AVG	35.6	37.3	23.8	28.5	35.0	27.2	28.4	36.7	26.1	34.5	32.4
AR	4.92	4.12	8.24	6.28	5.16	7.08	7.16	3.24	7.8	4.88	5.36

手法である。そのため、M5の方が fault 密度という量的変数の予測に向いている可能性がある。

5.2 RQ2: バージョンをまたがる fault 密度予測モデルの評価

バージョン A の各データセットを用いて fault 密度予測モデルを構築し、バージョン B の各データセットの fault-prone を予測した際の精度結果を表 5 に示す。ほぼ全てのデータセットにおいて、Recall が 25% を超える予測モデルが存在しており、また、平均で 35% を超える Recall 値を示すモデル構築手法が存在する。

また、各 fault 密度予測モデル構築手法の比較結果を図 3 に示す。バージョン間での fault 密度予測においても、RQ1 と同様に M5 が最も高い精度を示した。また RQ1 の場合に比べて M5 が有意差を示す手法が 3 種類から 5 種類へ増加しており、バージョン間での予測において M5 の精度が他手法と比較した場合に特に高くなるということが分かった。

更に、RQ1 と RQ2 の予測モデルの関係を調べるために、RQ1 と RQ2 の平均順位の相関係数を計算した結果、0.86 という強い相関を示した。つまり、fault 密度による fault-prone モジュール予測においても、過

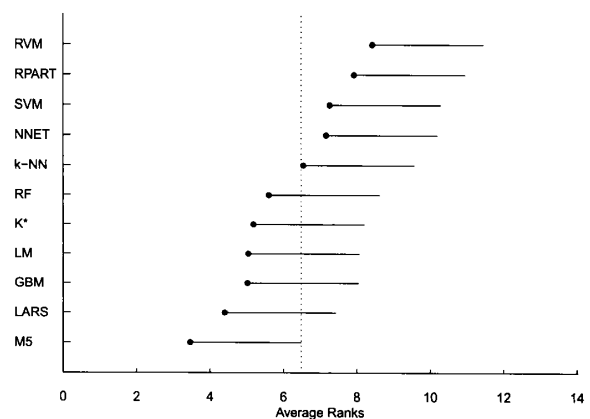


図 3 RQ2 の fault 密度予測時の Demšar's significance diagram

Fig. 3 Demšar's significance diagram in RQ2 using fault density as an objective variable.

去のバージョンにおける交差検証法を利用することにより、バージョンをまたいだ fault-prone モジュール予測のためのモデル構築手法の選択に役立つ可能性があることを示している。

6. む す び

本論文の目的はテスト工数考慮時に高い精度を示す fault-prone モジュール判別モデルの決定であり、その目的のために二つの RQ を立て、25 個のオープンソースプロジェクトを用いて検証を行った

RQ1 では 11 種類の fault 密度予測モデルを比較した。その結果、M5 が最も高い精度を示した。また、M5 をそれ以外の手法と比較した結果、平均順位の低位 3 手法 (RPART, RVM, SVM) に対して統計的に有意な差が見られた。

RQ2 では、過去のバージョンを用いた場合の 11 種類の fault 密度予測モデルを比較した。その結果、同一のバージョンを用いた交差検証法よりも精度は下がるものの、バージョン間においても M5 の精度が最も高く、平均順位の低位 5 手法に対して統計的に有意な差が見られた。また同一バージョンによる交差検証法での予測精度の平均順位とバージョン間の予測精度の平均順位には強い相関があった。

RQ1, RQ2 の実験結果を比較すると、順位が大きく変化した手法は存在せず (LM において RQ1 に比べ RQ2 では平均順位が 1.58 下がったのが最大)、またどちらの結果でも M5 の性能が最も高かった。これらの結果を受けて、本論文では、fault 密度予測モデルに M5 を用いることを推奨する。ただし、予測を実施する組織内で収集されたプロジェクトデータが十分に蓄積されている場合には、そのデータを用いて交差検証を行い、最も精度の高かった手法を採用することもある。

今回の結果から、実開発環境における fault 密度予測モデル適用へのおおよその道筋が示された。ただし、一部のプロジェクトでは、予測手法にかかわらず高い精度が得られなかった。今後の課題としては、高い予測精度が得られないプロジェクトについて、その特徴や原因を分析することが挙げられる。

謝辞 本研究の一部は、日本学術振興会科学研究費補助金 (若手 A: 課題番号 24680003) による助成を受けた。

文 献

- [1] N.E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Trans. Softw. Eng.*, vol.26, no.8, pp.797-814, 2000.
- [2] A. Mockus and D.M. Weiss, "Predicting risk of software changes," *Bell Labs Tech. J.*, vol.5, no.2,

- pp.169-180, 2000.
- [3] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," *Proc. Int'l Conf. on Softw. Eng. (ICSE'06)*, pp.452-461, 2006.
- [4] T.M. Khoshgoftaar, A.S. Pandya, and D.L. Lanning, "Application of neural networks for predicting program faults," *Annals of Software Engineering*, vol.1, no.1, pp.141-154, 1995.
- [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.," *Proc. Int'l Conf. on Softw. Eng. (ICSE'06)*, pp.413-422, 2006.
- [6] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," *Proc. Int'l Workshop on Mining Software Repositories (MSR'06)*, pp.119-125, 2006.
- [7] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Trans. Softw. Eng.*, vol.31, no.2, pp.380-391, 2005.
- [8] V.R. Basili, L.C. Briand, and W.L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol.22, no.10, pp.751-761, 1996.
- [9] L.C. Briand, V.R. Basili, and C.J. Hetmanski, "Developing interpretable models with optimized set reduction for identifying high-risk software components," *IEEE Trans. Softw. Eng.*, vol.19, no.11, pp.1028-1044, 1993.
- [10] T.M. Khoshgoftaar and E.B. Allen, "Modeling software quality with classification trees," *Recent Advances in Reliability and Quality Engineering*, pp.247-270, World Scientific, Singapore, 1999.
- [11] J.C. Munson and T.M. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Trans. Softw. Eng.*, vol.18, no.5, pp.423-433, 1992.
- [12] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Trans. Softw. Eng.*, vol.22, no.12, pp.886-894, 1996.
- [13] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," *IEEE Trans. Softw. Eng.*, vol.32, no.2, pp.69-82, 2006.
- [14] A.R. Gray and S.G. MacDonell, "Software metrics data analysis - exploring the relative performance of some commonly used modeling techniques," *Empirical Software Engineering*, vol.4, no.4, pp.297-316, 1999.
- [15] F. Xing, P. Guo, and M.R. Lyu, "A novel method for early software quality prediction based on support vector machine," *Proc. Int'l Symposium on Software Reliability Engineering (ISSRE'05)*, pp.213-222, 2005.
- [16] G.J. Pai and J. Bechta Dugan, "Empirical analysis

- of software fault content and fault proneness using bayesian methods," *IEEE Trans. Softw. Eng.*, vol.33, no.10, pp.675-686, 2007.
- [17] O. Mizuno and T. Kikuno, "Prediction of fault-prone software modules using a generic text discriminator," *IEICE Trans. Inf. & Syst.*, vol.E91-D, no.4, pp.888-896, April 2008.
- [18] T.M. Khoshgoftaar, N. Seliya, and N. Sundares, "An empirical study of predicting software faults with case-based reasoning," *Software Quality Control*, vol.14, no.2, pp.85-111, 2006.
- [19] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Softw. Eng.*, vol.34, no.4, pp.485-496, 2008.
- [20] T. Mende and R. Koschke, "Effort-aware defect prediction models," *Proc. European Conf. on Software Maintenance and Reengineering (CSMR'10)*, pp.109-118, 2010.
- [21] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A.E. Hassan, "Revisiting common bug prediction findings using effort aware models," *Proc. Int'l Conf. on Software Maintenance (ICSM'10)*, pp.1-10, 2010.
- [22] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," *Proc. Int'l Conf. on Softw. Eng. (ICSE'08)*, pp.181-190, 2008.
- [23] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Softw. Eng.*, vol.31, no.10, pp.897-910, 2005.
- [24] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," *Proc. Int'l Conf. on Softw. Eng. (ICSE'05)*, pp.284-292, 2005.
- [25] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Annals of Statistics*, vol.32, pp.407-499, 2004.
- [26] C.M. Bishop, "The relevance vector machine," *Advances in Neural Information Processing Systems 12*, pp.652-658, MIT Press, 2000.
- [27] J.G. Cleary and L.E. Trigg, "K*: An instance-based learner using an entropic distance measure," *Proc. Int'l Conf. on Machine Learning (ICML'95)*, pp.108-114, 1995.
- [28] N.F. Schneidewind and H.-M. Hoffmann, "An experiment in software error data collection and analysis," *IEEE Trans. Softw. Eng.*, vol.5, no.3, pp.276-286, 1979.
- [29] E. Arisholm, L.C. Briand, and E.B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *J. Systems and Software*, vol.83, no.1, pp.2-17, 2010.
- [30] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Trans. Softw. Eng.*, vol.26, no.7, pp.653-661, 2000.
- [31] S. Matsumoto, Y. Kamei, A. Monden, K. Matsumoto, and M. Nakamura, "An analysis of developer metrics for fault prediction," *Proc. Int'l Workshop on Predictor Models in Softw. Eng. (PROMISE'07)*, pp.18:1-18:9, 2010.
- [32] Y. Kamei, E. Shihab, B. Adams, A. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol.39, no.6, pp.757-773, 2013.
- [33] E. Shihab, Z.M. Jiang, W.M. Ibrahim, B. Adams, and A.E. Hassan, "Understanding the impact of code and process metrics on post-release defects: A case study on the Eclipse project," *Proc. Int'l Symposium on Empirical Softw. Eng. and Measurement (ESEM'10)*, pp.4:1-4:10, 2010.
- [34] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Softw. Eng.*, vol.31, no.4, pp.340-355, 2005.
- [35] R Core Team, "R: A language and environment for statistical computing," R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0. <http://www.R-project.org>
- [36] Classification and R. Training, "caret". <http://caret.r-forge.r-project.org/>

付 録

データセット詳細

実験にて使用したデータセットについて、バージョン、メトリックスの収集期間を表 A・1、表 A・2 に示す。メトリックス収集期間と fault 各プロジェクトがリリースされる頻度を考慮して決定された。

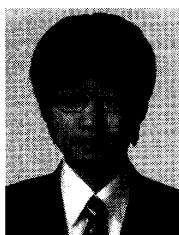
表 A.1 バージョン A 概要
Table A.1 Summary of version A (older).

プロジェクト	バージョン	メトリクス収集期間	fault 収集期間
Clam AntiVirus	0.96	2009/10/02 - 2010/03/31	2010/03/31 - 2010/09/27
eCos	2.0	2002/11/21 - 2003/05/20	2003/05/20 - 2006/05/04
Exim Internet Mailer	4.63	2006/02/01 - 2006/07/31	2006/07/31 - 2007/01/27
freedesktop	7.7	2009/09/22 - 2009/12/21	2009/12/21 - 2010/03/21
Ganymede	1.0pre1	2000/10/03 - 2001/01/31	2001/01/31 - 2001/05/31
Helma	1.3.1	2003/03/01 - 2003/08/28	2003/08/28 - 2004/02/24
Hibernate Core	3.6.1	2011/01/03 - 2011/02/02	2011/02/02 - 2011/03/04
HylaFAX	4.2.0	2003/08/17 - 2004/08/16	2004/08/16 - 2005/08/16
iptables	1.3.0	2003/08/22 - 2005/02/12	2005/02/12 - 2006/08/06
KOffice	2.1	2009/08/22 - 2009/11/20	2009/11/20 - 2010/02/18
NetBSD	4.0	2007/06/22 - 2007/12/19	2007/12/19 - 2008/06/16
OpenBSD	4.5	2009/01/31 - 2009/05/01	2009/05/01 - 2009/07/30
OpenCms	7.0.0	2007/01/06 - 2007/07/05	2007/07/05 - 2008/01/01
openNMS	1.7.0	2008/10/17 - 2009/01/15	2009/01/15 - 2009/04/15
OTRS	2.3	2008/01/24 - 2008/07/22	2008/07/22 - 2009/01/18
PostgreSQL	8.2	2006/06/05 - 2006/12/02	2006/12/02 - 2007/05/31
ProFTPD	1.3.2	2008/08/09 - 2009/02/05	2009/02/05 - 2009/08/04
Samba	3.2.0	2008/03/03 - 2008/07/01	2008/07/01 - 2008/10/29
Scilab	5.2.0	2009/06/18 - 2009/12/15	2009/12/15 - 2010/06/13
Spring Security	2.0.0	2007/10/17 - 2008/04/14	2008/04/14 - 2008/10/11
squid	2.4	2000/09/21 - 2001/03/20	2001/03/20 - 2001/09/16
Wine	1.1.0	2007/12/30 - 2008/06/27	2008/06/27 - 2008/12/24
XDoclet	1.2beta2	2002/09/29 - 2002/12/28	2002/12/28 - 2003/03/28
XFree86	4.0	1999/09/10 - 2000/03/08	2000/03/08 - 2000/09/04
XORP	1.0	2004/01/10 - 2004/07/08	2004/07/08 - 2005/01/04

表 A.2 バージョン B 概要
Table A.2 Summary of version B (newer).

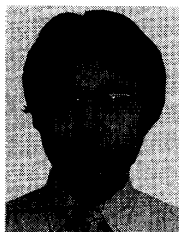
プロジェクト	バージョン	メトリクス収集期間	fault 収集期間
Clam AntiVirus	0.97	2010/08/11 - 2011/02/07	2011/02/07 - 2011/08/06
eCos	3.0	2008/10/01 - 2009/03/30	2009/03/30 - 2011/08/25
Exim Internet Mailer	4.68	2007/03/03 - 2007/08/30	2007/08/30 - 2008/02/26
freedesktop	7.8	2009/12/28 - 2010/03/28	2010/03/28 - 2010/06/26
Ganymede	1.0	2001/02/06 - 2001/06/06	2001/06/06 - 2001/10/04
Helma	1.4.0	2003/10/04 - 2004/04/01	2004/04/01 - 2004/09/28
Hibernate Core	3.6.2	2011/02/07 - 2011/03/09	2011/03/09 - 2011/04/08
HylaFAX	4.3.0	2005/05/18 - 2006/05/18	2006/05/18 - 2007/05/18
iptables	1.4.0	2006/06/30 - 2007/12/22	2007/12/22 - 2009/06/14
KOffice	2.2	2010/02/22 - 2010/05/23	2010/05/23 - 2010/08/21
NetBSD	5.0	2008/10/31 - 2009/04/29	2009/04/29 - 2009/10/26
OpenBSD	4.6	2009/07/20 - 2009/10/18	2009/10/18 - 2010/01/16
OpenCms	8.0.0	2010/11/09 - 2011/05/08	2011/05/08 - 2011/10/13
openNMS	1.8.0	2010/03/09 - 2010/06/07	2010/06/07 - 2010/09/05
OTRS	2.4	2009/01/23 - 2009/07/22	2009/07/22 - 2010/01/18
PostgreSQL	8.3	2007/08/05 - 2008/02/01	2008/02/01 - 2008/07/30
ProFTPD	1.3.3	2009/08/28 - 2010/02/24	2010/02/24 - 2010/08/23
Samba	3.3.0	2008/09/28 - 2009/01/26	2009/01/26 - 2009/05/26
Scilab	5.3.0	2010/06/18 - 2010/12/15	2010/12/15 - 2011/06/13
Spring Security	3.0.0	2009/06/25 - 2009/12/22	2009/12/22 - 2010/06/20
squid	2.5	2002/03/29 - 2002/09/25	2002/09/25 - 2003/03/24
Wine	1.2	2010/01/17 - 2010/07/16	2010/07/16 - 2011/01/12
XDoclet	1.2beta3	2003/03/20 - 2003/06/18	2003/06/18 - 2003/09/16
XFree86	4.1.0	2000/12/04 - 2001/06/02	2001/06/02 - 2001/11/29
XORP	1.1	2004/10/15 - 2005/04/13	2005/04/13 - 2005/10/10

(平成 24 年 10 月 10 日受付, 25 年 3 月 18 日再受付)



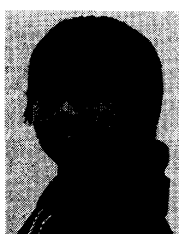
小林 寛武

平 21 九大・工・電気情報卒，平 24 同大学院システム情報科学府情報知能工学専攻修士課程了。在学中，マルチエージェントシステムにおける情報検索に関する研究に従事。



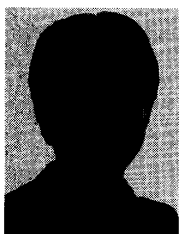
戸田 航史 (正員)

平 16 阪大・基礎工・卒，平 21 奈良先端科学技術大学院大学情報科学研究科博士後期課程了。平 24 より福岡工業大学情報工学科助教，博士（工学）。ソフトウェアメトリクスの研究に従事。情報処理学会，IEEE 各会員。



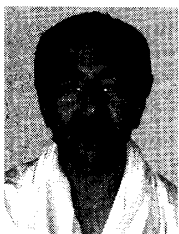
亀井 靖高 (正員)

平 17 関西大・総合情報卒，平 21 奈良先端科学技術大学院大学情報科学研究科博士後期課程了。同年日本学術振興会特別研究員（PD）。平 22 カナダ Queen's 大学博士研究員。平 23 九州大学大学院システム情報科学研究院助教，博士（工学）。ソフトウェアメトリクス，マイニングソフトウェアリポジトリの研究に従事。情報処理学会，IEEE 各会員。



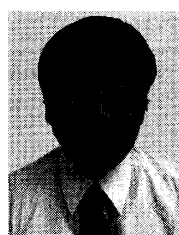
門田 暁人 (正員)

平 6 名大・工・電気卒，平 10 奈良先端科学技術大学院大学情報科学研究科博士後期課程了。同年同大学助手。平 16 同大学助教授。平 19 同大学准教授。平 15～16 Auckland 大学客員研究員。博士（工学）。ソフトウェアメトリクス，ソフトウェアプロテクション，ヒューマンファクタの研究に従事。情報処理学会，日本ソフトウェア科学会，IEEE，ACM 各会員。



峯 恒憲 (正員)

昭 62 九大・工・情報工学卒，平元同大学院総合理工学研究科情報システム学専攻修士課程了，平 4 同大学院同専攻博士後期課程単位取得の上退学。同年同大学教養部講師。平 6 同大学理学部物理学科講師，平 8 同大学院システム情報科学研究科（現，研究院）助教授（現 准教授），現在に至る。博士（工学）。自然言語処理，情報検索，マルチエージェントシステム，P2P ネットワーク，ソフトウェア開発プロセス等に関する研究に従事。1993 情報処理学会論文賞受賞。情報処理学会，言語処理学会，ACM，IEEE Computer Society 各会員。



鵜林 尚靖 (正員)

昭 57 広島大・理・数学卒，平 11 東京大学大学院総合文化研究科広域科学専攻広域システム科学系博士課程了。博士（学術）。昭 57～平 15（株）東芝に勤務。平 14～15 芝浦工業大学システム工学部非常勤講師。平 15 九州工業大学情報工学部助教授，平 22 九州大学大学院システム情報科学研究院教授，現在に至る。平成 15 年度情報処理学会山下記念研究賞受賞。ソフトウェア工学，プログラミング言語モデルに興味をもつ。日本ソフトウェア科学会，情報処理学会，ACM，IEEE-CS 各会員。