開発者に着目した Fault-Prediction 技術

高井 康勢
(株) 日立製作所
yasunari.takai.kb@hitachi.com

加藤 正恭
(株) 日立製作所
tadahisa.kato.en@hitachi.com

三部 良太
(株) 日立製作所
ryota.mibe.mu@hitachi.com

林 晋平
東京工業大学 情報理工学院
hayashi@c.titech.ac.jp

小林 隆志
東京工業大学 情報理工学院
tkobaya@c.titech.ac.jp

要旨

ソフトウェア開発では，開発の早い段階でフォールト（バグ）を見つけて修正することが重要となる．そのため，潜在的なフォールトの場所を予測する Fault-Prediction 技術に注目が集まっている．

Fault-Prediction 技術では，ファイルやモジュールごとに，潜在的なフォールトの存在確率を表す「潜在フォールト率」を算出する．算出した潜在フォールト率が高いファイルやモジュールを重点的にテストやレビューすることで，開発の早い段階でのフォールト発見が可能となる．

近年利用が進んでいる Fault-Prediction 技術では，静的解析ツールで検出できるコーディング規約違反や既知のフォールトの情報を基に，潜在フォールト率を算出する．しかしながら，開発の立ち上げ直後は，潜在フォールト率の算出に利用するコーディング規約違反やフォールトの情報が少ないため，従来技術では潜在フォールト率を算出できない．

そこで本研究では，「開発者」が過去に携わったプロジェクトで作り込んだ指摘やフォールトなどの情報を基に，ファイルごとの潜在フォールト率を算出する Fault-Prediction 技術を提案・評価した．本技術では，開発の立ち上げ直後でも潜在フォールト率を算出できる．

企業内のある 1 プロジェクトで提案手法を評価したところ，本手法で算出した潜在フォールト率が高い上位 25％のファイルに，全フォールトの約 6 割が含まれることが分かった．提案手法により，潜在フォールト率が高いファイルから順にテストやレビューを実施することで，開発の早い段階でのフォールトの発見・修正が可能となる．

1. はじめに

近年，大規模で複雑なソフトウェアを，高品質に素早く開発することが求められている．短い期間の中で品質の高いソフトウェアを作るためには，早期に効率よくフォールトを見つけ，修正による手戻りを最小限にすることが重要となる．このようなニーズに対して，潜在的なフォールトの場所を予測する，Fault-Prediction 技術に注目が集まっており，実プロジェクトでの利用も進んでいる．

Fault-Prediction 技術は，ソフトウェアの中から，潜在的なフォールトが存在する可能性が高い部分（モジュール・ファイル・クラス・メソッド・行など）を予測する技術 [1] である．具体的には，モジュールやファイルなどのソフトウェアの部分ごとに，潜在的なフォールトの存在確率を表す「潜在フォールト率」を算出する．Fault-Prediction 技術で算出した潜在フォールト率が高い部分を重点的にレビューやテストすることで，効率よくフォールトを見つけられる．

近年特に利用が進んでいる Fault-Prediction 技術には，既知のフォールトや静的解析ツール [2][3] で検出できるコーディング規約違反 [4][5] などの指摘を利用したものがある．これらの技術では，予測対象のプロジェクトにおける既知のフォールトや指摘を基に潜在フォールト率

を算出する．そのため，開発立ち上げ直後のプロジェクトなどで，既知のフォールトや指摘が少ない場合，潜在フォールト率を算出できないという課題がある．

この課題を解決するため，本研究では，ソフトウェアの品質に影響を与える「開発者」に着目し，開発者が過去に携わった既存のプロジェクトで作り込んだフォールトや指摘などの情報を基にした Fault-Prediction 技術を提案する．提案手法は，既存のプロジェクトにおけるフォールトや指摘を利用するため，立ち上げ直後のプロジェクトであっても利用可能である．

以降，2. では，関連する研究について説明する．3. では，提案手法について説明する．4. では，評価方法，評価対象，及び，結果を説明し，5. で考察を述べる．最後に，6. でまとめと今後の課題について述べる．

2. 関連研究

2.1. Fault-Prediction 技術

1971 年には，ソースコードの行数を指標とする Fault-Prediction 技術の研究 [6] が始まっており，以降，循環的複雑度 [7] などの指標を複数組み合わせた研究 [8] がおこなわれてきた．また，1990 年ごろからは，オブジェクト指向言語を対象とした研究 [9] なども行われている．

近年では，フォールトの分布や修正頻度を利用した研究 [10][11][12] で成果が出ており，実プロジェクトでの活用が進んでいる．代表的な方法は，「既知のフォールトがより多く存在するファイルに，より多くの潜在的なフォールトがある」と予測するものである．この方法では，ファイルごとに，「当該ファイルが原因になったフォールトの数」を集計して潜在フォールト率とする．

さらに，フォールトと静的解析ツールの指摘の関係があるという研究成果 [13][14] に基づき，指摘の分布や修正頻度を利用した研究 [15][16][17] もある．代表的な方法は，「指摘をより多く受けたファイルに，より多くのフォールトがある」と予測するものである．この方法では，まず，開発中のファイルに checkstyle[2] や FindBugs™[*1][3] などの静的解析ツールを適用して指摘を取得する．次に，ファイルごとの指摘数を集計して潜在フォールト率とする．

[*1] FindBugs™ とそのロゴは，the University of Maryland の登録商標です．

2.2. 開発者の特徴

ソースコードなどの成果物の品質には，開発者の影響があることが知られている．例えば，Avgustinov ら [18] は，開発者ごとに受ける指摘に違いがあることを示した．Matsumoto ら [19] は，開発者の特徴を表すメトリクスを提案し，開発者ごとに違いがあることを示した．

3. 提案手法

提案手法である「開発者に着目した Fault-Prediction 技術」は，開発の立ち上げ直後であっても利用できる Fault-Prediction 技術である．本手法は，ソフトウェアの開発者に着目し，開発者が過去に携わった既存のプロジェクトで作り込んだフォールトや指摘などの実績情報を基に，予測対象プロジェクトのファイルごとにフォールトが含まれる可能性を潜在フォールト率として算出することを特徴とする．

以降，3.1 にて実績情報を定義したのち，3.2 で開発者に着目した Fault-Prediction 技術について説明する．

3.1. 実績情報

本研究では，実績情報を「開発者のフォールトの作り込みやすさを直接的／間接的に表すメトリクス」と定義する．例えば，「開発行あたりの平均フォールト数」や「開発行あたりの平均指摘数」は，実績情報である．他にも，Matsumoto ら [19] が提案した「コミットあたりの平均フォールト混入率」など，開発者ごとに測定可能で，かつ，Fault-Prediction 技術で利用可能なメトリクスも実績情報である．

ここで，本論文で実績情報として利用した開発行あたりの平均フォールト数と開発行あたりの平均指摘数の定義を示す．

定義. 開発者 d の開発行あたりの平均フォールト数:Pf*(d)*

$$\begin{aligned}\mathrm{Pf}(d) &= \text{開発者 } d \text{ の開発行あたりの平均フォールト数} \\ &= \frac{\mathrm{FL}(d)}{\mathrm{DL}(d)}\end{aligned}$$

$$\mathrm{FL}(d) = \text{開発者 } d \text{ が開発した行に含まれるフォールト数} = \sum_{\text{開発者 } d \text{ が開発した行 } l} \mathrm{F}(l)$$

$$\mathrm{F}(l) = \text{行 } l \text{ が原因となったフォールトの数}$$

$$\mathrm{DL}(d) = \text{開発者 } d \text{ が開発した行数}$$

定義. 開発者 d の開発行あたりの平均指摘数:Pv*(d)*

$$\mathrm{Pv}(d) = \text{開発者 } d \text{ の開発行あたりの平均指摘数} = \frac{\mathrm{V}(d)}{\mathrm{DL}(d)}$$

$$\mathrm{V}(d) = \text{開発者 } d \text{ が開発した行に含まれる指摘数}$$

$$\mathrm{DL}(d) = \text{開発者 } d \text{ が開発した行数}$$

3.2. 開発者に着目した Fault-Prediction 技術

提案手法である開発者に着目した Fault-Prediction 技術では，以下に示す定義に基づいて潜在フォールト率を算出する．算出した潜在フォールト率が高いファイルを順にレビューやテストすることで，開発の早い段階でフォールトを見つけて修正することが可能となる．

定義. 実績情報に基づく潜在フォールト率:FR*(f)*

$$\mathrm{FR}(f) = \sum_{\text{ファイル } f \text{ の行 } l} \mathrm{P}(\mathrm{Dev}(l))$$

$$\mathrm{Dev}(l) = \text{行 } l \text{ の開発者}$$

$$\mathrm{P}(d) = \text{既存のプロジェクトにおける開発者 } d \text{ の実績情報}$$

尚，$\mathrm{P}(d)$ は，3.1 で定義した $\mathrm{Pf}(d)$，$\mathrm{Pv}(d)$ 等を表す．

4. 評価

開発の立ち上げ直後において，提案手法で算出した潜在フォールト率が高いファイルを重点的にレビューやテストすることで，フォールトを効率的に見つけられるか評価した．

4.1. 評価方法

具体的な評価方法は以下の通りである．

1. 開発がある程度進んでおり，尚且つ，同じ開発者が従事しているプロジェクトを 2 つ用意し，一方を開発者が過去に関わった既存のプロジェクト，もう一方を開発の立ち上げ直後である予測対象のプロジェクトとみなす．
2. 既存のプロジェクトを基に開発者の実績情報を取得し，その実績情報を基にして予測対象のプロジェクトの開発の立ち上げ直後における各ファイルの潜在フォールト率 FR(*f*) を算出する．
3. FR(*f*) が高い順にファイルをレビューやテストした時に発見可能なフォールト数を算出する．
4. ランダムにファイルをレビューやテストした時に発見可能なフォールト数を算出し，3. と比較する．

尚，評価は，3.2 で示した開発者の実績情報に基づく潜在フォールト率算出において，$\mathrm{P}(d)$ として $\mathrm{Pf}(d)$ を用いた方法と，$\mathrm{Pv}(d)$ を用いた方法の 2 通りで行った．以降，前者を FPf, 後者を FPv と呼ぶ．

表 1. 評価対象ソフトウェアの概要

開発者数	9 名
コミット数	17,000
規模（行数）	約 70KLOC
規模（ファイル数）	214
開発期間	約 1 年
フォールト数	224
指摘数	38,187

4.2. 評価対象

企業内の Java ソフトウェアのプロジェクトを対象に，提案手法を評価した．評価対象ソフトウェアの概要は，表 1 の通りである．尚，フォールトには，単体テストで見つけたフォールトと，結合テストで見つけたフォールトが含まれている．

本評価では，1 つのプログラムを機能の観点で 2 つに分割し，2 つのプロジェクトに見立てて評価した．両モジュールの概要は，表 2，表 3 の通りである．以降，両モジュールを，それぞれ「モジュール A」「モジュール B」と呼ぶ．

FPf, FPv 双方に対して，モジュール A を既存のプロジェクト，モジュール B を予測対象のプロジェクトとみなした評価と，その逆の評価を実施した（表 4）．以降，各

表 2. モジュール A の概要

規模（行数）	約 61KLOC
規模（ファイル数）	164
フォールト数	202
指摘数	33,226

表 3. モジュール B の概要

規模（行数）	約 9KLOC
規模（ファイル数）	50
フォールト数	22
指摘数	4,961

評価を，表中の「#」に記載した名称で呼ぶ．

4.3. 評価結果

B→A@FPf 及び B→A@FPv の結果を図 1 に，A→B@FPf 及び A→B@FPv の結果を図 2 に示した．両グラフの横軸は，提案手法で算出した潜在フォールトが高い順にファイルを並べたものであり，縦軸は，累積フォールト率を表している．累積フォールト率は，全フォールト数に対する，潜在フォールト率が高い順にテストやレビューを行った際に発見可能なフォールトの割合である．つまり，「潜在フォールト率が高いファイル上位 x 個の中に，全フォールトの y% のフォールトが含まれているか」を表している．例えば，図 1 では，潜在フォールト率が高いファイル上位 40 個をテストやレビューすることで，最大で，全フォールトの約 6 割のフォールトを発見できることを表している．

また，図 1 と図 2 における，潜在フォールト率が高いファイル上位 25%，50%，75% までの累積フォールト率を表 5 に示した．尚，理論上の最大値は，既知のフォールトが多い順にファイルを並べたものであり，また，ランダムサンプリングは，10 回ファイルをランダムに並べた際の，累積フォールト率の平均である．

図 1，図 2，及び，表 5 をもとに，FPf 及び FPv の結果について述べる．

FPf では，潜在フォールト率が上位 25% のファイルに，B→A@FPf で 64%，A→B@FPf で 55% のフォールトが含まれていた．どちらも，25% のファイルをランダムに選んだ場合と比較すると，倍以上のフォールトを含んでいる．また，潜在フォールト率が上位 50%，75% の場合においても，ファイルをランダムに選んだ場合より，多くのフォールトが含まれていた．

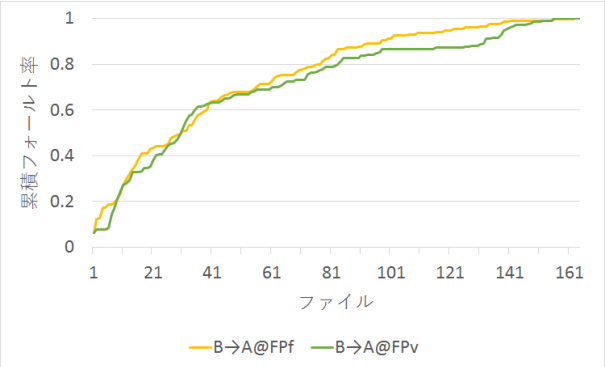

図 1. B→A@FPf 及び B→A@FPv における潜在フォールトの予測結果

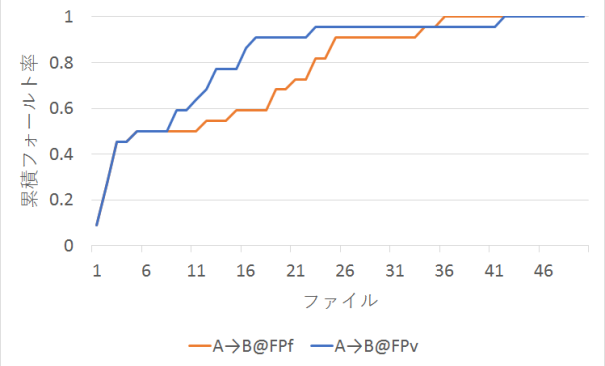

図 2. A→B@FPf 及び A→B@FPv における潜在フォールトの予測結果

FPv では，潜在フォールト率が上位 25% のファイルに，B→A@FPv で 63%，A→B@FPv で 68% のフォールトが含まれていた．どちらも，25% のファイルをランダムに選んだ場合と比較すると，倍以上のフォールトを含んでいる．また，潜在フォールト率が上位 50%，75% の場合においても，ファイルをランダムに選んだ場合より，多くのフォールトが含まれていた．

FPf と FPv を比較すると，潜在フォールト率が上位

表 4. 評価内容

#	対象モジュール		実績情報
	既存のプロジェクト	予測対象のプロジェクト	
B→A@FPf	モジュール B	モジュール A	FPf
B→A@FPv	モジュール B	モジュール A	FPv
A→B@FPf	モジュール A	モジュール B	FPf
A→B@FPv	モジュール A	モジュール B	FPv

表 5. 評価結果

#	上位 25%	上位 50%	上位 75%
A@ファイル数	41	82	123
A@理論上の最大値	81%	100%	100%
B→A@FPf	64%	84%	96%
B→A@FPv	63%	79%	87%
A@ランダムサンプリング	28%	53%	77%
B@ファイル数	13	25	38
B@理論上の最大値	100%	100%	100%
A→B@FPf	55%	91%	100%
A→B@FPv	68%	95%	95%
B@ランダムサンプリング	20%	46%	66%

75% の場合においては，FPf の方が多くのフォールトを含んでいるが，それ以外の場合では，一定の傾向はみられなかった．

5. 考察

5.1. 提案手法の有用性

4.3 で示した通り，立ち上げ直後のプロジェクトであっても，提案手法を利用し，潜在フォールト率が上位 25% のファイルを重点的にレビューやテストすることで，約 6 割のフォールトを発見できる可能性がある．

大規模で複雑なソフトウェアを高品質に素早く開発するためには，早期に効率的にフォールトを発見することが重要となる．しかしながら，従来手法は，立ち上げ直後のプロジェクトでの利用が難しい．一方，提案手法では，立ち上げ直後のプロジェクトであっても，効率的なフォールト発見を実現できると言える．

5.2. FPf と FPv の比較

4.3 で示した通り，本評価では，FPf と FPv の間に優位な差を見出すことはできなかった．但し，特にモジュール B は，ファイル数が 50 ファイルとそれ程多くないため，測定誤差による影響が大きくでている可能性がある．

5.3. 従来手法との比較

従来手法と提案手法を比較しながら，提案手法であれば，開発の立ち上げ直後であっても潜在フォールト率を算出できる理由を考察する．

図 3 は，時間経過に伴う，既知フォールト，指摘，実績情報の変化と，従来手法及び提案手法での潜在フォールト率の算出可否を表している．既存のプロジェクトも予測対象のプロジェクトも，立ち上げ直後には既知のフォールトが存在しておらず，また，指摘も少ない．開発が進むと，まず指摘が多くなり，さらに開発が進むと，既知のフォールトも多くなる．尚，図中の網掛け部分は，従来手法，提案手法で潜在フォールト率を算出できる部分を表している．

従来手法，提案手法ともに，予測に利用する既知のフォールトや指摘の情報（「予測基情報」と呼ぶ）が少ないと，潜在フォールト率の算出ができない．一方で，従来手法と提案手法には，既存のプロジェクトの予測基情報を利用できるか否かの点で差がある．

従来手法は，予測対象のファイルにおけるフォールトや指摘の情報を予測基情報として利用する．既存のプロジェクトで作成したファイルと予測対象のプロジェクトで作成したファイルは異なるため，既存のプロジェクトの予測基情報を予測対象のプロジェクトで利用することができない．それに対し，提案手法では，予測対象の開発者における実績情報を予測基情報として利用する．そのため，同じ開発者が既存のプロジェクトと予測対象のプロジェクトを担当していれば，既存のプロジェクトで得られる予測基情報を予測対象のプロジェクトで利用できる可能性がある．

本評価より，既存のプロジェクトで得られる予測基情報を予測対象のプロジェクトで利用できることが分かった．

6. まとめと今後の課題

近年，ソフトウェアに存在するフォールトの発見を支援できる Fault-Prediction 技術に注目が集まり，実プロジェクトでの利用が進んでいる．しかしながら，従来の Fault-Prediction 技術では，フォールトの予測の基になる情報が少ない立ち上げ直後のプロジェクトでは，利用が難しいという課題がある．そこで本研究では，開発者が過去に関わった既存のプロジェクトを分析し，開発者ごとの実績情報を得たうえで予測の基になる情報として利用し，開発の立ち上げ直後でも利用できる Fault-Prediction 技術を提案した．

提案手法を企業内のある 1 プロジェクトで評価したところ，実績情報として「開発行あたりの平均フォールト数」と「開発行あたりの平均指摘数」を利用した場合，フォールトが存在する確率が高いと予測した上位 25% のファイルに全フォールトの約 6 割が含まれるという結果を得られた．この結果は，ランダムに 25% のファイルを選択した際に含まれるフォールトの割合（20%〜30%）と比較すると，2 倍程度大きい．つまり，提案手法により算出したフォールトの存在確率が高いファイルを重点的にレビューやテストすることで，立ち上げ直後のプロジェクトであっても，早期のフォールト発見を支援できると言える．

尚，本研究では 1 プロジェクトのみを利用して評価したため，他のプロジェクトを含めた評価が今後の課題である．また，今回利用した 2 つの実績情報以外の実績情報でも評価し，予測精度向上の可能性を探る．

参考文献

[1] Rakesh Kumar and D. L. Gupta. Software fault prediction : A review. In *International Journal of Computer Science and Mobile Computing*, Vol. 4, pp. 250–254, Sep 2015.

[2] checkstyle checkstyle 8.18. `http://checkstyle.sourceforge.net/`.

[3] Findbugs - find bugs in java programs. `http://findbugs.sourceforge.net/`.

[4] MISRA-C 研究会. 組込み開発者におくる MISRA - C:2004—C 言語利用の高信頼化ガイド. 日本規格協会, 10 2006.

[5] Pep 8 – style guide for python code — python.org. `https://www.python.org/dev/peps/pep-0008/`.

[6] Fumio Akiyama. An example of software system debugging. In *Proceedings of IFIP Congress*, pp. 353–359, 1971.

[7] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, pp. 308–320, Dec 1976.

[8] V. Y. Shen, , S. M. Thebaut, and L. R. Paulsen.

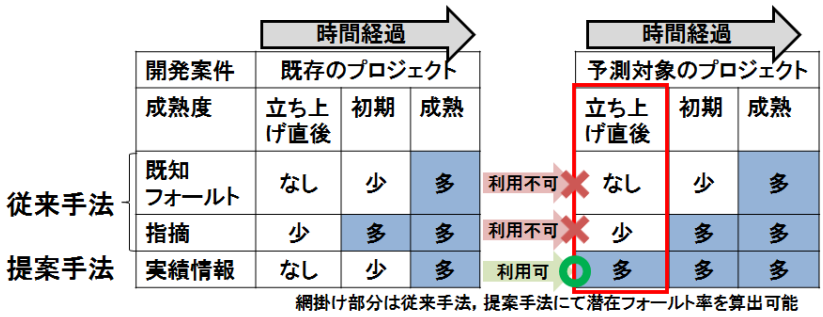

図 3. 既知フォールト，指摘，実績情報の変化と潜在フォールト率の算出可否

Identifying error-prone software - an empirical study. *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4, pp. 317–324, April 1985.

[9] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Proceedings of 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 31–41, May 2010.

[10] Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl T. Barr, and Premkumar T. Devanbu. Bugcache for inspections: hit or miss? In *Proceedings of FSE*, pp. 322–331. ACM, 2011.

[11] Chirs Lewis and Rong Ou. Bug prediction at google — google engineering tools. `http://google-engtools.blogspot.com/2011/12/bug-prediction-at-google.html`, 12 2011.

[12] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller. Predicting faults from cached history. In *Proceedings of 29th International Conference on Software Engineering (ICSE 2007)*, pp. 489–498, May 2007.

[13] Nathaniel Ayewah and William Pugh. A report on a survey and study of static analysis users. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, (DEFECTS 2008), pp. 1–5, 2008.

[14] Foyzur Rahman, Sameer Khatri, Earl T. Barr, and Premkumar Devanbu. Comparing static bug finders and statistical prediction. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, pp. 424–434, 2014.

[15] Wojciech Basalaj. Correlation between coding standards compliance and software quality. In *White paper, Programming Research Ltd*, 2008.

[16] C. Boogerd and L. Moonen. Evaluating the relation between coding standard violations and faultswithin and across software versions. In *Proceedings of 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR 2009)*, pp. 41–50, May 2009.

[17] Y. Takai, T. Kobayashi, and K. Agusa. Software metrics based on coding standards violations. In *Proceedings of 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp. 273–278, Nov 2011.

[18] P. Avgustinov, A. I. Baars, A. S. Henriksen, G. Lavender, G. Menzel, O. d. Moor, M. Schafer, and J. Tibble. Tracking static analysis violations over time to capture developer characteristics. In *Proceedings of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE 2015)*, Vol. 1, pp. 437–447, May 2015.

[19] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, Ken-ichi Matsumoto, and Masahide Nakamura. An analysis of developer metrics for fault prediction. In *Proceedings of the 6th In-*

ternational Conference on Predictive Models in Software Engineering (PROMISE 2010), pp. 18:1–18:9, 2010.