情報処理学会論文誌 Vol. 50 No. 7 1716–1724 (July 2009)

Fault-proneモジュール判別におけるテスト工数割当てとソフトウェア信頼性のモデル化

柿元 健[†1] 門田暁人[†2] 亀井靖高[†2]
枩本真佑[†2] 松本健一[†2] 楠本真二[†1]

ソフトウェアの信頼性確保を目的として，fault の有無を推定するモデル（fault-prone モジュール判別モデル）が数多く提案されている．しかし，どのようにテスト工数を割り当てるのかといった fault-prone モジュール判別モデルの判別結果の利用方法についての議論はほとんどされておらず，信頼性確保の効果は不明確であった．そこで，本論文では，fault の有無の判別，テスト工数の割当て，ソフトウェア信頼性の関係のモデル化を行い，TEAR（Test Effort Allocation and software Reliability）モデルを提案する．TEAR モデルにより，与えられた総テスト工数の枠内で，ソフトウェア信頼性が最大となるような（モジュールごとの）テスト工数割当ての計画立案が可能となる．TEAR モデルを用いてシミュレーションを行った結果，推定される判別精度が高い，もしくは，fault 含有モジュールが少ない場合には，fault-prone モジュールに多くのテスト工数を割り当てた方がよく，推定される判別精度が低い，もしくは，fault 含有モジュールを多く含む場合には，判別結果に基づいてテスト工数を割り当てるべきではないことが分かった．

Modeling of Test Effort Allocation and Software Reliability in Fault-prone Module Detection

Takeshi Kakimoto,[†1] Akito Monden,[†2]
Yasutaka Kamei,[†2] Shinsuke Matsumoto,[†2]
Ken-ichi Matsumoto[†2] and Shinji Kusumoto[†1]

Various fault-prone detection models have been proposed to improve software reliability. However, while improvement of prediction accuracy was discussed, there was few discussion about how the models shuld be used in the field, i.e. how test effort should be allocated. Thus, improvement of software reliability by fault-prone module detection was not clear. In this paper, we proposed TEAR (Test Effort Allocation and software Reliability) model that represents the relationship among fault-prone detection, test effort allocation and software reliability. The result of simulations based on TEAR model showed that greater test effort should be allocated for fault-prone modules when prediction accuracy was high and/or when the number of faulty modules were small. On the other hand, fault-prone module detection should not be use when prediction accuracy was small or the number of faulty modules were large.

1. はじめに

ソフトウェア開発プロジェクトにおいて，信頼性の向上，テスト工数削減を目的としてソフトウェアモジュールの欠陥（fault）の有無を推定する（fault-prone モジュール判別）ための研究がさかんに行われてきた[11),16)]．モジュールの欠陥の有無の推定には，fault-prone モジュール判別モデルと呼ばれる，モジュールにおいて計測された規模，複雑度，継承の深さなどのメトリクス値に基づいた，線形判別分析，ロジスティック回帰分析，ニューラルネット，分類木などによるモデル（fault-prone モジュール判別モデル）を用いた判別がよく用いられている[3),8),9),12),14),17)]．

しかし，判別方法が数多く提案され，実データを用いた判別精度の評価も行われているにもかかわらず，その多くは開発現場で採用されていない．その原因の 1 つは，判別による実質的な効果（信頼性の向上やテスト工数削減）が不明確なためである．fault-prone モジュール判別によって期待される効果は，fault を含むと判定されたモジュール（fault-prone モジュール）に重点的にテスト工数を割り当てることで，(1) 信頼性の確保と (2) 無駄なテスト工数の削減の両立である．ただし，このような効果を推定し，テスト工数を割り当てる方法は従来知られていないため，たとえ精度の良い判別結果が得られたとしても，その有用性の程度は不明であった．

そこで本論文では，fault-prone モジュール判別の結果に基づいたテスト工数の割当てと，割当ての結果得られるソフトウェア信頼性の関係をモデル化する．このモデルを TEAR（Test Effort Allocation and software Reliability）モデルと呼ぶ．TEAR モデルにより，与えられた総テスト工数の枠内でソフトウェア信頼性が最大となるような（モジュールごと

†1 大阪大学大学院情報科学研究科
Graduate School of Information and Science Technology, Osaka University
†2 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

の）テスト工数割当ての計画立案が可能となる．

TEAR モデルでは，ソフトウェア信頼性の指標として，ソフトウェアの潜在 fault 数に対する発見された fault 数の割合である潜在 fault 発見率を用い，テスト工数の割当てとソフトウェア信頼性の関係をモデル化する．一般に，fault-prone モジュール判別結果はある程度の誤りを含むため，テスト工数の割当ての決定は単純ではない．仮に，fault を含まないと判別されたモジュール（non-fault-prone モジュール）にいっさいテスト工数を割り当てなかった場合，判別が完全に正しければ問題ないが，判別誤りを含む場合には，判別を誤ったモジュールの fault は発見されないためソフトウェアの信頼性の低下を招くこととなる．そこで，TEAR モデルでは，non-fault-prone モジュールに対しても一定のテスト工数を割り当てることを想定し，パラメータとして指定可能とする．

本論文では，さらに，TEAR モデルに基づいたシミュレーションを行うことで，TEAR モデルを用いた fault-prone モジュール判別を効果的に活用するテスト工数の割当ての計画立案の方法について明らかにする．シミュレーションは，(1) 判別精度の指標の 1 つで適合率と再現率の調和平均である F1 値[5),14)] が異なる場合，(2) 潜在 fault を実際に含んでいるモジュールである fault 含有モジュールの割合が異なる場合，について行う．シミュレーションにより，高いソフトウェア信頼性の確保に適したテスト工数の割り当て方を明らかにすることで，fault-prone モジュール判別を効果的に活用可能となることが期待される．

以降，2 章では，TEAR モデルについて説明する．3 章では，TEAR モデルを用いたシミュレーションとその結果について述べ，4 章ではシミュレーションの結果について議論する．5 章では，関連研究について述べ，最後に 6 章で本論文のまとめを述べる．

2. テスト工数割当てとソフトウェア信頼性のモデル（TEAR モデル）

2.1 テスト工数の割当てに関するポリシ

本論文では，テスト工程としてウォーターフォール型開発の単体テスト，統合テストを対象とする．そして，パッケージやサブシステム単位ではなく，モジュール（ファイル）単位での予測を対象とするため，Moser ら[13)] と同様に，fault の数や確率の予測ではなく，fault の有無は 2 値判別されるものとする．また，判別結果に対し，開発現場では，次のポリシに基づいてテスト工数の割当てを行うとする．

Fault を含むと判別されたモジュール（fault-prone モジュール）により多くのテスト工数を割り当てる．逆に，fault を含まないと判別されたモジュール（non-fault-prone モジュール）には少しのテスト工数しか割り当てない．具体的には，fault-prone モジュールには，non-fault-prone モジュールの r 倍のテスト工数が割り当てられる．ここで，r は 1 以上の実数である．

この条件の下では，判別精度が低いほど，実際には fault を含まないモジュールにより多くのテスト工数を割り当てたり，fault が存在するモジュールに少ししか工数を割り当てなかったりすることとなり，信頼性の低下を招く．

2.2 1 モジュールにおけるテスト工数と fault 発見のモデル

各モジュールでは，2.1 節のテスト工数割当てによって，ある確率で fault が発見される．一般に，モジュールに fault が含まれている場合，より多くのテスト工数を費やすほど fault の発見確率は高まる．ただし，fault が含まれていない場合は，費やしたテスト工数にかかわらず fault は発見されない．ここでは，1 モジュールにおける fault の有無，テスト工数の割当て，fault 発見確率の関係をモデル化する．

本論文では，1 モジュールにおけるモデルとして，比較的単純なモデルである指数関数モデルを採用する．指数関数モデルは，ソフトウェア全体の信頼性を推定するためのソフトウェア信頼度成長モデル（SRGM: Software Reliability Growth Model）[2),18)] でもよく用いられており，テスト工数と fault 発見の関係を簡潔に表現できる．

あるモジュールにおける fault の発見確率を式 (1) で定義する．

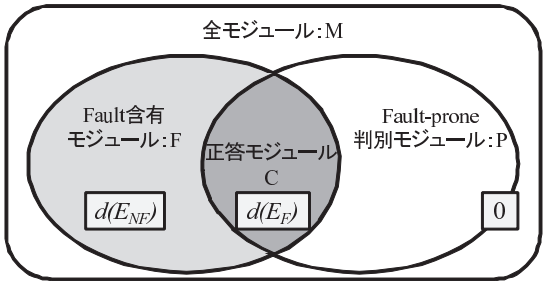
$$d(E) = \begin{cases} 1 - e^{-zE} & \text{fault を含む} \\ 0 & \text{fault を含まない} \end{cases} \tag{1}$$

ここで，$d(E)$ は，あるモジュールに工数 E を割り当てたときの fault 発見確率，z（小文字）は，単位工数あたりに当該モジュールの fault が発見される確率（厳密には，確率そのものではなく，確率を決定する定数）である．この指数関数モデルでは，fault は費やしたコスト（テスト工数）に対して一様の確率で発見される．

2.3 ソフトウェア全体におけるテスト工数割当てと信頼性のモデル

式 (1) のモデルは，1 モジュールにおけるテスト工数と fault の発見との関係を表したモデルである．このモデルを，ソフトウェア全体における（各モジュールへの）テスト工数割当てとソフトウェア信頼性の関係を表すモデルへと拡張する．ここでは，ソフトウェア信頼性の指標として，ソフトウェアの潜在 fault 数に対する発見された fault 数の割合である潜在 fault 発見率を用いる．

モデル化において，**図 1** のように，ソフトウェアの全モジュールの集合を M，実際に



E_F: Fault-prone モジュールに割り当てるテスト工数
E_{NF}: Non-fault-prone モジュールに割り当てるテスト工数

図 1　モジュールの関係と fault 発見確率
Fig. 1　Relationship between modules and the percentages of discoverable faults.

fault を含むモジュールである fault 含有モジュールの集合を F，fault-prone と判別されたモジュールである fault-prone モジュールの集合を P，fault-prone モジュールと判別された fault 含有モジュールである正答モジュールの集合を C とする．また，それぞれのモジュール数を，全モジュール数 N_M，fault-prone モジュール数 N_P，fault 含有モジュール数 N_F，正答モジュール数 N_C とする．

Fault-prone モジュール 1 つあたりに費やすテスト工数を E_F，non-fault-prone モジュール 1 つあたりに費やすテスト工数を E_{NF} とすると，各モジュールにおける fault 発見確率は次のようになる．

- Fault 含有モジュールのうち fault-prone と判別された正答モジュール（図 1 の C）：$d(E_F)$
- Fault 含有モジュールのうち non-fault-prone と判別されたモジュール（図 1 の $F-C$）：$d(E_{NF})$
- Fault を含まないモジュール（図 1 の $M-F$）：0

したがって，それぞれの fault 発見確率で fault が発見されるモジュールの割合は，次のようになる．

- $d(E_F)$ で fault が発見されるモジュールの割合：N_C/N_F
- $d(E_{NF})$ で fault が発見されるモジュールの割合：$1-(N_C/N_F)$

そこで，全モジュールの fault 発見率 $D(E_F, E_{NF})$ は式 (2) で表される．

$$\begin{aligned} D(E_F, E_{NF}) &= d(E_F) \times \frac{N_C}{N_F} + d(E_{NF}) \times \left(1 - \frac{N_C}{N_F}\right) \\ &= \left(1-e^{-Z \times E_F}\right) \times \frac{N_C}{N_F} + \left(1-e^{-Z \times E_{NF}}\right) \times \left(1 - \frac{N_C}{N_F}\right) \end{aligned} \tag{2}$$

ここで，Z（大文字）は，当該ソフトウェアの残存 fault 数に対する単位工数あたりに発見される fault 数の期待値である．

式 (2) の E_F，E_{NF} は，2.1 節のテスト工数割当てポリシから次の関係で表される．

$$E_{all} = E_F \times N_P + E_{NF} \times (N_M - N_P) \tag{3}$$

$$E_F = E_{NF} \times r \tag{4}$$

ここで，E_{all} は全テスト工数であり，fault-prone モジュールには non-fault-prone モジュールの r 倍のテスト工数が割り当てられるものとする．

式 (3) と式 (4) の連立方程式を E_F と E_{NF} について解くと

$$E_F = \frac{E_{all} \times r}{N_P(r-1) + N_M} \tag{5}$$

$$E_{NF} = \frac{E_{all}}{N_P(r-1) + N_M} \tag{6}$$

が得られる．

式 (2) に，式 (5)，式 (6) を代入すると，式 (7) が得られる．

$$\begin{aligned} D(E_F, E_{NF}) = &\left(1 - e^{-\frac{E_{all} \times r}{N_P(r-1)+N_M} Z}\right) \times \frac{N_C}{N_F} \\ &+ \left(1 - e^{-\frac{E_{all}}{N_P(r-1)+N_M} Z}\right) \times \frac{N_F - N_C}{N_F} \end{aligned} \tag{7}$$

式中の各記号と意味を**表 1** に示す．

また，判別精度指標として，正答モジュール数 N_C の代わりに，fault-prone モジュール判別でよく用いられる F1 値を用いて式 (7) を表すと，式 (8) となる．

$$\begin{aligned} D(E_F, E_{NF}) = &\left(1 - e^{-\frac{E_{all} \times r}{N_P(r-1)+N_M} Z}\right) \times \frac{F1}{2}\left(1 + \frac{N_P}{N_F}\right) \\ &+ \left(1 - e^{-\frac{E_{all}}{N_P(r-1)+N_M} Z}\right) \times \left(1 - \frac{F1}{2}\left(1 + \frac{N_P}{N_F}\right)\right) \end{aligned} \tag{8}$$

ここで，F1 値は次式で表される．

表 1　式中の記号の意味
Table 1　The meaning of symbols in equations.

記号	意味
$D(E_F, E_{NF})$	ソフトウェア全体の fault 発見率
$d(E)$	1 モジュールあたりの fault 発見確率
E_{all}	全テスト工数
E_F	Fault-prone 判別モジュールに費やすテスト工数
E_{NF}	Non-fault-prone 判別モジュールに費やすテスト工数
N_M	全モジュール数
N_P	Fault-prone 判別モジュール数
N_F	Fault 含有モジュール数
N_C	正答モジュール数
r	Fault-prone 判別モジュールに割り当てる工数の倍率
Z	単位工数あたりの fault 発見率
z	単位工数あたりに該当モジュールにおいて fault が発見される確率

$$F1 = \frac{2N_C}{N_F + N_P} \tag{9}$$

本論文では，式 (7) と式 (8) をテスト工数割当てとソフトウェア信頼性のモデル（TEAR モデル）とする．

3. シミュレーション

3.1 シミュレーションの概要

本論文では，限られたテスト工数の枠内でソフトウェア信頼性を最大とするためのテスト工数割当ての計画立案の方法を明らかにするため，TEAR モデルに基づいたシミュレーションを行う．シミュレーションは，(1) 判別精度（F1 値）が異なる場合，(2) 潜在 fault を実際に含んでいるモジュール（fault 含有モジュール）の割合が異なる場合，について行う．

(1)，(2) のシミュレーションを実施するにあたっては，non-fault-prone モジュールに対して fault-prone モジュールに割り当てるテスト工数の倍率（r）を変化させて，信頼性への影響を調べる．判別精度が高い場合，non-fault-prone モジュールにはテスト工数をほとんど割り当てなくてもよいが，判別精度が低い場合には，non-fault-prone モジュールにもテスト工数を割り当てないと多くの fault を見逃すこととなるため，最適な r を知ることがテスト現場において必要となるためである．

TEAR モデルへの入力として，fault 含有モジュール数 N_F，または，全モジュールに対するその割合（fault 含有モジュール率 $P_{FPi} = N_F/N_M$）を与える必要がある．これらの値は，本来は fault-prone モジュール判別実施時には未知であるが，本シミュレーションでは，過去の開発事例から推定できるものと仮定する．シミュレーション (1) では $P_{FPi} = 0.2$ を採用し，シミュレーション (2) では P_{FPi} を 0.2 刻みで変化させた．また，fault-prone モジュール数 N_P についても何らかの値を与える必要がある．N_P は fault-prone モジュール判別モデルとモデル構築に用いたデータセットの fault 含有モジュール率（P_{FPi}）によって決まるため，シミュレーションにおいて妥当な値は存在しない．そのため，本論文のシミュレーションでは，実際に fault を含むモジュール数だけ fault-prone モジュールと予測した場合である N_P/N_M の値が fault 含有率 P_{FPi} と同一の値となるように，つまり，N_P は N_F と同じ値とした．

また，全モジュール数 N_M，全テスト工数 E_{all}，単位工数あたりに発見される fault の割合 Z については，定数としてシミュレーションを行った．全モジュール数については，NASA/WVU で公開されているデータ[15] を，全テスト工数については，情報処理推進機構（IPA）ソフトウェア・エンジニアリング・センターで収集されたデータ[6] を参考にして，以下の値に決定した．

- 全モジュール数：$N_M = 1500$（個）
- 総テスト工数：$E_{all} = 1700$（人時）
- 単位工数あたりに発見される fault の割合：$Z = 0.5$

全モジュール数と総テスト工数の出典は異なるが，ソフトウェアの規模（ソースコード行数）がともに数万ステップのデータを出典としている．また，$Z = 0.5$ の場合，1 モジュール単位で見ると，1 モジュールでの単位工数あたりに fault が発見される確率は付録 A.1 のように 39%である．

Fault-prone モジュール判別における代表的な評価指標としては，適合率（Precision），再現率（Recall），F1 値があげられる[5]．これらの評価指標のうち，適合率と再現率はトレードオフの関係にあるため，本シミュレーションでは，これらの調和平均の値である F1 値を採用し，判別精度として用いる．

3.2 シミュレーション (1)：推定される判別精度が異なる場合

3.2.1 方　　法

シミュレーション (1) の目的は，推定される判別精度が異なる場合の各モジュールに割り当てるテスト工数の決定方法を明らかにすることである．そのために，判別精度（F1 値）ごとに non-fault-prone モジュールに対する fault-prone モジュールに割り当てるテスト工数の倍率 r を変化させ，ソフトウェア全体の fault 発見率 D の変化を確かめる．

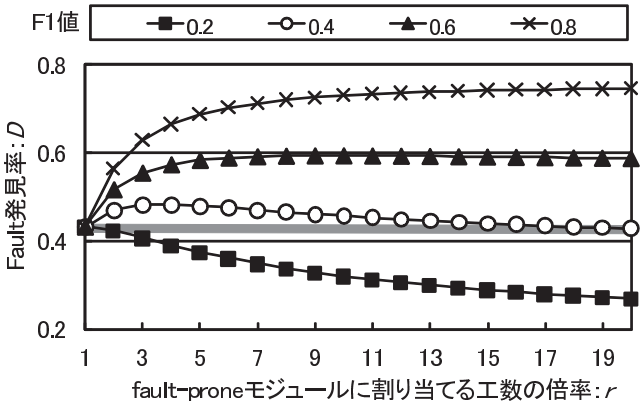


図 2　シミュレーション (1)：F1 値ごとの fault 発見率
Fig. 2　Simulation 1: Fault detection rate in each F1 value.

シミュレーション (1) では，パレートの法則を考慮して fault 含有モジュール率 P_{FPi} を 0.2，fault-prone と判定するモジュールの割合（N_P/N_M）は P_{FPi} と同じ値（0.2）とし，正答モジュール数 N_C は F1 値にあわせて変化させた．

3.2.2　結　　果

シミュレーション (1) の結果を**図 2** に示す．グラフの横軸は non-fault-prone モジュールに対する fault-prone モジュールに割り当てるテスト工数の倍率 r の値を，縦軸はソフトウェア全体の fault 発見率 D を示す．r が 1 の場合は，fault-prone モジュールと non-fault-prone モジュールに同数の工数を割り当てる．すなわち，fault-prone モジュール判別を行わない場合のソフトウェア全体の fault 発見率である．グラフ中の灰色の線は，この fault-prone モジュール判別を行わない場合のソフトウェア全体の fault 発見率 D の値を示す．

シミュレーション (1) の結果，判別精度が低い（F1 値が 0.2）場合には，fault-prone モジュール判別を行うことで r が 1 のとき（fault-prone モジュール判別を行わない場合）よりもソフトウェア全体の fault 発見率 D が低下している．すなわち，fault-prone モジュール判別結果に基づいた工数割当てを行うことで，行わない場合よりも fault 発見率を低下させており，判別結果に基づいてテスト工数を割り当てるべきではないことを示している．

一方，F1 値が 0.4，0.6 の場合には，fault-prone モジュールにテスト工数を多く割り当てる（r の値を大きくする）ことで，fault-prone モジュール判別を行わない場合よりもソフトウェア全体の fault 発見率 D は増加している．しかし，fault-prone モジュールにより多くのテスト工数を割り当てるにつれて，fault 発見率 D の増加の度合いは次第に小さくなっている．そして，さらに多くのテスト工数を割り当てると，fault 発見率 D は低下に転じている．また，グラフの表示範囲内には表れていないが，F1 値が 0.8 の場合も，同様の結果となっている．これは，fault-prone モジュールに割り当てるテスト工数を増加させることは，non-fault-prone モジュールに割り当てるテスト工数を減少させることになるため，non-fault-prone と判定されたモジュールのうち，実際には fault 含有モジュールであった（判別を誤った）モジュールの fault 発見率が低下するためである．

3.3　シミュレーション (2)：Fault 含有モジュール率が異なる場合

3.3.1　方　　法

シミュレーション (2) の目的は，fault 含有モジュール率 P_{FPi} が異なる場合の，各モジュールに割り当てるテスト工数の決定方法を明らかにすることである．そのために，fault 含有モジュール率ごとに fault-prone モジュールに割り当てるテスト工数の倍率 r を変化させ，ソフトウェア全体の fault 発見率 D の変化を確かめる．

シミュレーション (2) では，F1 値を（比較的精度の高いケースを想定して）0.6 とし，シミュレーション (1) と同様に正答モジュール数 N_C は fault 含有モジュール率にあわせて変化させ，fault-prone と判定するモジュールの割合（N_P/N_M）は fault 含有率 P_{FPi} と同一の値とした．

3.3.2　結　　果

シミュレーション (2) の結果を**図 3** に示す．グラフの横軸は r の値を，縦軸はソフトウェア全体の fault 発見率 D を示す．グラフ中の灰色の線は，fault-prone モジュール判別を行わない場合の fault 発見率 D の値を示す．

シミュレーション (2) の結果，fault 含有モジュール率が高い（fault 含有モジュール率が 0.6，0.8）場合には，fault-prone モジュール判別を行うことで，fault-prone モジュール判別結果に基づいた工数割当てを行わない場合よりも fault 発見率を低下させており，判別結果に基づいてテスト工数を割り当てるべきではないことを示している．

一方，fault 含有モジュール率が低い場合には，fault-prone モジュールにテスト工数を多く割り当てることで，fault-prone モジュール判別結果に基づいた工数割当てを行わない場合よりも fault 発見率 D は増加している．そして，シミュレーション (1) と同様に，fault-prone モジュールにより多くのテスト工数を割り当てるにつれて，fault 発見率 D の増加の度合いは次第に小さくなっていき，さらに多くのテスト工数を割り当てると，fault 発見率 D は低下に転じている．

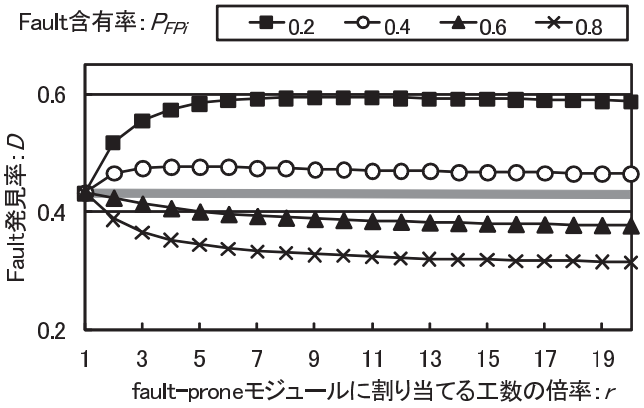


図 3　シミュレーション (2)：Fault 含有モジュール率ごとの fault 発見率
Fig. 3　Simulation 2: Fault detection rate in each fault module rate.

4. 考　　察

シミュレーション (1) の結果より，推定される fault-prone モジュール判別の判別精度が低い場合には，fault-prone モジュール判別を行うことが fault 発見率を下げることが示唆された．本論文のシミュレーションの場合では，$P_{FPi} = 0.2$，つまり，全モジュールの 20%が fault を含む場合，F1 値 $= 0.2$ 程度では判別の効果がなく，F1 値 $= 0.4$ 程度以上ならば，信頼性向上の効果が見込めることが示唆された．ただし，シミュレーション (2) の結果から，fault 含有モジュール率 P_{FPi} が高くなるほど，要求される F1 値も大きくなることが示唆され，本論文のシミュレーションの場合では，$P_{FPi} = 0.6$，つまり，全モジュールの 60%が fault を含む場合，F1 値 $= 0.6$ であっても判別の効果はない．これらのことから，テスト工数割当てを行う者は，過去のプロジェクトの事例などから P_{FPi} と F1 値を推定してから TEAR モデルにあてはめ，fault-prone モジュール判別結果を採用するかどうかを決定する必要がある．

シミュレーション (2) の結果からは，fault 含有モジュール率 P_{FPi} が高い場合には，F1 値が比較的高い場合であっても，判別の効果がないことが示唆された．このことから，fault-prone モジュール判別に基づくテスト工数割当ては，ソフトウェア中の残存バグが少なくなった時点で実施すべきであるといえる．一般に，ソフトウェアの単体テストの段階では，数多くのバグが検出されることから fault-prone モジュール判別はあまり役立たないと考えられる．一方，結合テストや総合テストの段階では，残存バグ数が少なくなり fault 含有モジュール率 P_{FPi} が小さくなるため，fault-prone モジュール判別が役立つことが多くなると考えられる．

また，non-fault-prone モジュールに対して fault-prone モジュールに割り当てるテスト工数の倍率（r）は，推定される P_{FPi} と F1 値に応じて決定する必要があることが分かった．たとえば，本論文のシミュレーション (2) の結果においては，$P_{FPi} = 0.4$，F1 値 $= 0.6$ であれば，$r = 4$ が望ましいが，より fault 含有モジュール率が低いと期待される（$P_{FPi} = 0.2$）場合には，$r = 10$ 付近が望ましい．ただし，r を大きくすればするほど，推定される fault 含有モジュール率が誤っていた場合に低下する信頼性も大きくなることから，r を小さくしてローリスクローリターンでいくか，r を高くしてハイリスクハイリターンでいくかは，（過去のプロジェクトの事例などからの）推定される fault 含有モジュール率の確からしさに基づいてテスト管理者が決定する必要がある．

なお，fault-prone モジュール判別実施時には，判別精度や P_{FPi} は未知であるため，これらの値を事前に推定しておくことが重要である．判別精度は，fault-prone モジュール判別モデル構築に用いた過去のソフトウェア開発で計測，収集したモジュールのデータにおいて，交差検証法[1] によって判別を実施することで推定可能である．また，fault 含有モジュール率も同様に，過去に開発したモジュールのデータなどから推定可能であると考えられる．

TEAR モデルでは，厳密性よりもモデルの利便性を重視したため，各モジュールの fault 発見確率は一定とし，ソフトウェア全体としてパラメータ Z を設けている．より厳密なモデル化のためには，超指数型 SRGM のように，モジュールごとに規模などを考慮して異なるパラメータを与えることができるように拡張することが今後の課題となる．現状の TEAR モデルにおいてこの点を補う 1 つの方法は，テスト工数の割当てにおいて，予測結果に基づいて fault-prone モジュールに non-fault-prone モジュールの r 倍の工数を割り当てた後で，さらに，各モジュールの規模に応じてテスト工数を上乗せする，といった処理を行うことが考えられる．

5. 関 連 研 究

モジュールの fault の有無を推定するための fault-prone モジュール判別モデルは数多く提案されており，線形判別分析，ロジスティック回帰分析，ニューラルネット，分類木などがよく用いられている[3),9),11),14),16),17)]．さらに，サンプリング法[8)] や外れ値除去[12)] などのデータセットを操作する手法の併用も提案されている．これらの研究では，判別モデルの

性能の改善，もしくは比較を目的としており，信頼性の向上やテスト工数削減といった判別の効果については触れられておらず，fault-prone モジュール判別モデルの有用性は不明である．

Fault-prone モジュール判別におけるソフトウェア信頼性の向上を目的とした研究として，Khoshgoftaar ら[10]，および，Moser ら[13] の研究がある．これらの研究は判別誤り（特に第 2 種の誤り）の結果生じるコスト（工数）に注目し，判別誤りをできるだけ少なくすることで，信頼性を向上させる方法を提案している．具体的には，fault-prone モジュール判別モデルの出力値を fault-prone モジュールあるいは non-fault-prone モジュールと判定するための閾値を変化させることで，コストの最小化を試みている．これらの手法を用いることで，fault-prone モジュール判別をより効果的に活用できると考えられるが，判別誤りを完全になくすことはできないため，判別後に TEAR モデルを用いてテスト工数の割り当て方を決定することで，さらなる信頼性の向上が実現できると考えられる．

6. おわりに

本論文では，fault-prone モジュール判別の判別結果に基づいたテスト工数の割り当て方，割り当てた結果の効果を明らかにするために，fault の有無の判別，テスト工数の割当て，ソフトウェア信頼性（fault 発見率）をモデル化し，TEAR モデルを構築した．また，テスト工数割当てとソフトウェアの信頼性（fault 発見率）の関係をシミュレーションによって明らかにした．シミュレーションの結果，以下のことが分かった．

- 推定される判別精度が高い，もしくは，fault 含有モジュールが少ない場合には，fault-prone モジュールに多くのテスト工数を割り当てた方が高い信頼性が得られる．ただし，割り当てすぎるとソフトウェアの信頼性は低下する．
- 推定される判別精度が低い，もしくは，fault 含有モジュールを多く含む場合には，判別結果に基づいてテスト工数を割り当てない方が高い信頼性が得られる．

本論文では，単純なモデルである指数関数モデルを基に fault 発見率モデルを構築した．TEAR モデルにおいて，fault 発見に関するモデルは他のモデルに置き換え可能であるため指数関数以外のモデルについても検討すること，TEAR モデルを用いて実際にテスト工数の割当てを行い，TEAR モデルの効果を確認すること，適合率と再現率を変化させてシミュレーションを行うこと，および，TEAR モデルを fault の有無を確率で出力する fault-prone モジュール判別モデルに対応するように拡張することが今後の課題である．

謝辞　本研究の一部は，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．また，特別研究員奨励費（課題番号：20009220）の研究助成を受けて行われた．

参考文献

1) Browne, W.M.: Cross-validation methods, *Journal of Mathematical Psychology*, Vol.44, No.1, pp.108–132 (2000).
2) Goel, L.A. and Okumoto, K.: Time-dependent errordetection rate model for software reliability and other performance measures, *IEEE Trans. Reliability*, Vol.28, No.3, pp.206–211 (1979).
3) Gray, R.A. and MacDonell, G.S.: 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).
4) Gyimothy, T., Ferenc, R. and Siket, I.: 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).
5) Herlocker, L.J., Konstan, A.J., Terveen G.L. and Riedl, T.J.: Evaluating collaborative filtering recommender systems, *ACM Trans. Information Systems*, Vol.22, No.1, pp.5–53 (2004).
6) 独立行政法人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：ソフトウェア開発データ白書 2006—IT 企業 1400 プロジェクトの定量データで示す開発の実態，日経 BP 社，東京 (2006).
7) Kamei, Y., Monden, A. and Matsumoto, K.: Empirical evaluation of svm-based software reliability model, *Proc. 5th ACM-IEEE International Symposium on Empirical Software Engineering* (*ISESE2006*), Vol.2, pp.39–41 (2006).
8) 亀井靖高，枩本真佑，柿元　健，門田暁人，松本健一：Fault-Prone モジュール判別におけるサンプリング法適用の効果，情報処理学会論文誌，Vol.48, No.8, pp.2651–2662 (2007).
9) Khoshgoftaar, M.T. and Allen, B.E.: Modeling software quality with classification trees, *Recent Advances in Reliability and Quality Engineering*, pp.247–270, World Scientific (1999).
10) Khoshgoftaar, T.M., Yuan, X. and Allen, E.B.: Balancing Misclassification Rates in Classification-Tree Models of Software Quality, *Empirical Software Engineering*, Vol.5, No.4, pp.313–330 (2000).
11) Li, L.P., Herbsleb, J., Shaw, M. and Robinson, B.: Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc., *Proc. 28th International Conference on Software Engineering* (*ICSE'06*), pp.413–422 (2006).

12) Matsumoto, S., Kamei, Y., Monden, A. and Matsumoto, K.: Comparison of outlier detection methods on fault-proneness models, *Proc. 1st International Symposium on Empirical Software Engineering and Measurement* (*ESEM2007*), pp.461–463 (2007).
13) 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 International Conference on Software Engineering* (*ICSE'08*), pp.181–190 (2008).
14) Munson, C.J. and Khoshgoftaar, M.T.: The detection of fault-prone programs, *IEEE Trans. Softw. Eng.*, Vol.18, No.5, pp.423–433 (1992).
15) NASA IV&V Facility, Metrics Data Program. http://mdp.ivv.nasa.gov/
16) Ohlsson, N. and Alberg, H.: Predicting fault-prone software modules in telephone switches, *IEEE Trans. Softw. Eng.*, Vol.22, No.12, pp.886–894 (1996).
17) Pighin, M. and Zamolo, R.: A predictive metric based on discriminant statistical analysis, *Proc. 19th International Conference on Software Engineering* (*ICSE'97*), pp.262–270 (1997).
18) Ramamoorthy, V.C. and Bastani, B.F.: Software reliability-status and perspective, *IEEE Trans. Softw. Eng.*, Vol.8, No.4, pp.354–371 (1982).

付　　録

A.1　式 (1) の挙動

1 モジュールにおけるテスト工数と fault 発見確率のモデルである式 (1) において，前提条件を次のようにした場合

- あるモジュールに fault が含まれている．
- 単位工数（1 人時）あたりに当該モジュールで fault が発見される確率（z）$= 0.5$
- 要員は 1 人

$d(E)$ の変動は次のようになる．

1 時間 $d(E) = 0.39$：39%の確率でモジュールの fault が検出・除去される．
2 時間 $d(E) = 0.63$：63%の確率でモジュールの fault が検出・除去される．
3 時間 $d(E) = 0.78$：78%の確率でモジュールの fault が検出・除去される．
⋮

以上のように，工数を費やすほどモジュールの fault 発見確率が向上し，そのモジュールの信頼性は向上する．そして，工数を費やすにつれて fault 発見確率の向上度合いは小さくなる．また，単位工数（1 人時）あたりに当該モジュールで fault が発見される確率（z）によって，fault 発見確率は増減する．ただし，モジュールに fault が含まれていない場合には，いくら工数を費やしても fault 発見確率は向上しない．

表 2　式 (1) と SRGM の対応関係
Table 2　Relationship between equation (1) and SRGM.

本論文の式 (1)	SRGM
$d(E) = 1 - e^{-zE}$	$E(t) = a(1 - e^{-bt})$
$d(E)$：fault 発見確率	$E(t)$：期待発見 fault 数
E：工数	t：時間
	a：$t = 1$ における潜在 fault 数
z：単位工数あたりに当該モジュールで fault が発見される確率（を決定する定数）	b：単位時間あたりの fault 発見率（を決定する定数）

A.2　指数型 SRGM と式 (1) の比較

指数型 SRGM と式 (1) の fault を含む場合の関係は**表 2** のようになる．

SRGM，式 (1) ともに，かけたコストに対して発見される fault が算出されるが，SRGM はソフトウェア全体を対象としているのに対して，式 (1) は 1 モジュールを対象とする．また，本論文では，モジュールの fault 数は扱わず fault の有無のみを扱うため，SRGM の潜在 fault 数（a）にあたる変数は 0 か 1 の 2 値をとることになる．SRGM において $a = 1$ とした場合，「期待発見 fault 数」は「fault が発見される確率」と読み替えることができるため，本論文の式 (1) の $d(E)$ は「fault 発見確率」としている．そして，SRGM の単位時間あたりの fault 発見率 b は，式 (1) では工数に基づくため単位工数あたりに当該モジュールで fault が発見される確率 z としている．そのため，式 (1) は，fault の有無のみを扱うため指数型 SRGM の fault 発見数を fault 発見確率と読み替え，また，時間 t を工数 E と読み替えたものであるといえる．

(平成 20 年 7 月 30 日受付)
(平成 21 年 4 月 3 日採録)

柿元　　健（正会員）

平成 15 年神戸市立工業高等専門学校専攻科電気電子工学専攻修了．平成 20 年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修了．同年大阪大学情報科学研究科特任助教．博士（工学）．IT スペシャリストの育成，および，ソフトウェア信頼性，ソフトウェアコスト見積りの研究に従事．電子情報通信学会，IEEE 各会員

門田　暁人（正会員）

平成 6 年名古屋大学工学部電気学科卒業．平成 10 年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修了．同年同大学助手．平成 16 年同大学助教授．平成 19 年同大学准教授．平成 15〜16 年 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．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

亀井　靖高（学生会員）

平成 17 年関西大学総合情報学部卒業．平成 19 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学博士後期課程在籍．平成 20 年日本学術振興会特別研究員（DC2）採用，エンピリカルソフトウェア工学，特にソフトウェア信頼性の研究に従事．電子情報通信学会，IEEE 各会員．

松本　真佑（学生会員）

平成 18 年京都産業大学理学部卒業．平成 20 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学博士後期課程在籍．平成 21 年日本学術振興会特別研究員（DC2）採用，エンピリカルソフトウェア工学，特にソフトウェアメトリクスの研究に従事．電子情報通信学会，IEEE 各会員

松本　健一（正会員）

昭和 60 年大阪大学基礎工学部情報工学科卒業．平成元年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 5 年奈良先端科学技術大学院大学助教授．平成 13 年同大学教授．工学博士．エンピリカルソフトウェア工学，特に，プロジェクトデータ収集/利用支援の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員，IEEE Senior Member．

楠本　真二（正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 8 年同大学講師．平成 11 年同大学助教授．平成 14 年同大学情報科学研究科助教授．平成 17 年同大学教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．電子情報通信学会，IEEE，JFPUG，PM 各会員．