

バグモジュール予測を用いた テスト工数割り当て戦略のシミュレーション

中野 大輔 門田 暁人 亀井 靖高 松本 健一

ソフトウェア品質の確保とテストの効率化を目的として、バグモジュール予測方法が盛んに研究されている。しかし、予測結果を用いたテスト戦略(多数のモジュールに対するテスト工数の配分方法)については、従来、ほとんど研究されていない。本論文では、多数のテスト戦略の効果を評価するための、テストのシミュレーションモデルを提案する。シミュレーションモデルは、総テスト工数、テスト戦略、テスト対象のモジュールの複雑さ、及び、バグモジュール予測結果を入力とし、発見可能なバグ数を出力とする。ケーススタディとして、2つのオープンソース開発プロジェクト(Eclipse, Mylyn)を対象としたシミュレーションを行い、総テスト工数と有効な戦略との関係を明らかにした。

To date, various techniques for predicting fault-prone modules have been proposed; however, test strategies, which assign a certain amount of test effort to each module, have been rarely studied. This paper proposes a simulation model of software testing that can evaluate various test strategies. The simulation model estimates the number of discoverable faults with respect to the given test resources, the test strategy, complexity metrics of a set of modules to be tested, and the fault prediction results. Based on a case study of simulation applying fault prediction to two open source software (Eclipse and Mylyn), we show the relationship between the available test effort and the effective test strategy.

1 はじめに

近年、ビジネスサイクルの短期化とグローバル市場における競争の激化により、ソフトウェア開発の短期化、低コスト化が強く求められている。その一方で、大規模・複雑化するソフトウェアへの品質要求は高いため、ソフトウェアテストをできる限り効率的に行うことが必須となっている[20]。従来、テストの効率化を行うために、バグを含むモジュール(バグモジュール)の予測方法が盛んに研究されてきた^{†1}。開発者

は、予測結果に基づき、限られたテスト工数をバグモジュール(と予測されたもの)に重点的に割り当てることで、より少ない工数でより多くのバグの検出を目指す[9]。

しかし、従来研究では、主に予測精度向上への取り組みや新たなモデル化手法の提案などに留まっており、バグモジュール予測結果に基づいたテスト工数の割り当て方法についてはほとんど研究されておらず、バグモジュール予測の本来の目的であるテスト工数削減や品質向上確保の効果は明らかにされていない。

本論文では、バグモジュール予測結果に基づいたテスト工数割り当て方法の効果を評価することを目的とする。そのために、リサーチクエストン1では、代表的なテスト工数割り当て戦略をリストアップし、それらの優劣を「平均的なテスト工数の下でのバグ発見率」により比較する。また、リサーチクエストン2では、テスト工数を変化させたときの、最も効果の

Simulation of Effort Allocation Strategies in Software Testing Using Bug Module Prediction.

Daisuke Nakano, Akito Monden, Kenichi Matsumoto, 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.

Yasutaka Kamei, 九州大学システム情報科学研究所, Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University.

コンピュータソフトウェア, Vol.31, No.2 (2014), pp.118-128.

[研究論文] 2013年4月29日受付。

†1 本論文では、バグという用語を fault(モジュール中に

含まれる誤り)という意味で用いる。

高いテスト工数割り当て戦略を明らかにする。これらを実施するために、本論文では、与えられたテスト戦略により発見できるバグ数を推定するためのシミュレーションモデルを提案する。各リサーチクエションの詳細は次の通りである。

[RQ1] 最も効果の高いテスト戦略は何か: テスト戦略としては様々なものが考えられる。例えば、各モジュールのテスト工数を、バグ数 (の期待値) に比例させて割り当てる戦略が考えられる。他にも、規模に比例させる、複雑さ (サイクロマティック数) に比例させる、バグ密度^{†2} に比例させるといった戦略が考えられる。RQ1 では、ソフトウェア開発データ白書 [8] に基づく平均的なテスト工数 (32.7 人時/KLOC) のもとで、いずれのテスト戦略が最も多くのバグを発見できるのかを明らかにする。

どのようなテスト戦略が有効であるかは、利用可能な総テスト工数によって異なる可能性がある。そこで、次の RQ2 に答える。

[RQ2] 総テスト工数は有効なテスト戦略にどのような影響を与えるか: 総テスト工数を変化させたときの有効なテスト戦略について明らかにする。例えば、テスト工数に余裕がある場合、(バグ予測を行わずに) 全てのモジュールを網羅的にテストする方がより多くのバグを発見できる可能性がある。

続く 2 章では関連研究について述べる。3 章ではバグモジュール予測について説明する。4 章では、バグモジュール予測の結果を用いたテスト工数割り当てのシミュレーションについて示す。5 章でシミュレーション実験の概要を述べた後に、6 章に本論文で立てたりサーチクエションに対する実験結果を記す。最後に 7 章で本論文のまとめを述べる。

2 関連研究

従来、バグモジュール予測における予測精度向上への取り組みが報告されている。また、予測モデルの評価指標の提案 [3] [16] や、ソフトウェア開発企業での

バグモジュール予測の適用事例 [12] [14] [22] [26] が報告されている。文献 [14] では企業でのバグモジュール予測の効果を定性的に示した事例が報告されており、文献 [22] ではバグモジュール予測を自社の開発プロダクト、プロセスへ適用し、バグモジュール予測を機能テスト工程以前に完了させる事で、信頼性の確保に貢献するとしている。しかし、いずれの研究においてもどの程度のテスト工数削減が見込めるかについて、定量的な数値を示していない。

バグモジュール予測の結果に基づいたテスト方法に関する研究も少ないながら行われている [1] [7] [10] [18]。阿萬 [1] は、バグモジュール予測の結果に基づいて重点的にテストするモジュールを選択するという課題を、0-1 計画問題として定式化し、モジュール間の依存関係を考慮した重点モジュールの選択を可能としている。阿萬はテストすべきモジュールを選択することに焦点をあてているのに対し、本論文では、各モジュールへのテスト工数の配分方法を明らかにすることを目的としている点が異なる。

柿元ら [10] は、テスト工数と発見できるバグ数との関係をモデル化し、バグモジュール判別の予測結果に基づいたテスト工数割り当ての効果をシミュレーションにより評価している。ただし、柿元らの研究では、モジュールの規模や複雑さを考慮しておらず、全モジュールにおいてバグの発見のしやすさが一定であると仮定している。一方、本論文では、モジュール毎でバグ発見確率が異なるという事象をモデル化している。

林ら [7] と Monden [18] らは、バグ予測にかかるコストと、バグ予測の効果とのトレードオフを明らかにすることを目的として、バグの発見のしやすさがモジュール規模に反比例するという仮定に基づくテスト・シミュレーションモデルを提案し、テスト工数削減の効果を推定している。また、36 モジュールから成るソフトウェアを対象として、受け入れテストにおけるバグ予測の計画立案、メトリクス計測、バグ予測モデルの構築等に要したコストを明らかにしている。一方、本論文は、これらの研究の発展として、特にシミュレーションに焦点を当てており、バグの発見のしやすさがモジュールの複雑さ (サイクロマティック

^{†2} 単位行数あたりのバグ数。

数)に反比例すると仮定している点が異なる。サイクロマティック数はテストにおける分岐網羅を達成するために必要なテストケース数を表しており、バグの発見のしやすさをより正確に表現できると期待される。また、本論文では、数千モジュールから構成される2つの大規模システム (Eclipse, Mylyn) を対象としていること、ソフトウェア開発データ白書 [8] に基づいて、大規模開発を想定してシミュレーションモデルのパラメータを決定していること、及び、バグ数・バグ密度の予測に加えてバグの有無の予測を行った場合の効果の評価している点が異なる。

3 バグモジュール予測

バグモジュール予測では、ソフトウェアを構成する各モジュールに含まれるバグ数 [2], バグ密度 [16], または、バグの有無 [27] [24] の予測を行う。そのために、各モジュールの特性値 (規模, サイクロマティック数, ネストの深さ, 変更回数, 追加行数, 削除行数など) を説明変数とする予測モデルを構築する。モデルの構築のために、過去の開発の実績データを用いる。

3.1 モデル化手法

本論文では、バグモジュール予測のモデル化手法として広く用いられているロジスティック回帰分析, 重回帰分析, ランダムフォレスト法の3種類のモデル化手法を用いる。

ロジスティック回帰分析 (LR) は、2クラスの標本を判別する目的で用いられる。本論文では、バグの有無の判別 (バグを1個以上含むモジュールと含まないモジュールの判別) に用いる。ただし、判別結果を直接テスト戦略に用いるのではなく、判別値 (バグの有無の期待値) に基づいて、テスト工数の割り当てを行う。ロジスティック回帰分析の詳細については、文献 [9] を参照されたい。

重回帰分析は、数値を予測する目的で広く用いられる。詳細については、文献 [19] を参照されたい。本論文では、バグ数、及び、バグ密度の予測に用いる。

ランダムフォレスト法 [4] は、多数の決定木 (分類木、または、回帰木) を用いて集団学習を行う手法で

あり、Breiman によって提案された。学習データ (モデル構築用データ) にオーバーフィッティングしにくいという特長を持ち、バグ予測において最も有望なモデルの1つであることが知られている [13]。本論文では、バグの有無の判別、バグ数・バグ密度の予測に用いる。

3.2 評価指標

本論文では予測モデルの評価指標として Alberg Diagram [24] の AUC と CE Curve [3] の AUC を用いる。

Alberg Diagram とは、バグモジュール予測結果に基づいて、バグを含んでいる可能性の高い順に (判別値の大きな順に) モジュールを抽出した際に、実際にバグが存在するモジュールをどれだけ抽出できたかの割合をグラフ化したものである。図1に示すように横軸は予測値の高い順に抽出されたモジュールの個数をパーセントで表し、縦軸は実際にバグを含んでいたモジュールの数をパーセントで表す。AUC は曲線下面積の事であり、予測モデルの精度が高いほどグラフの形は左上に凸形状になり、AUC の値は上昇する。AUC の値域は [0,1] であり、1つの目安として 0.7 を超えることが求められる [13] [17]。

CE Curve の AUC はモジュールの規模を考慮した予測モデルの評価指標である。図2に示すように、横軸を「予測バグ密度の高い順にモジュールを抽出した

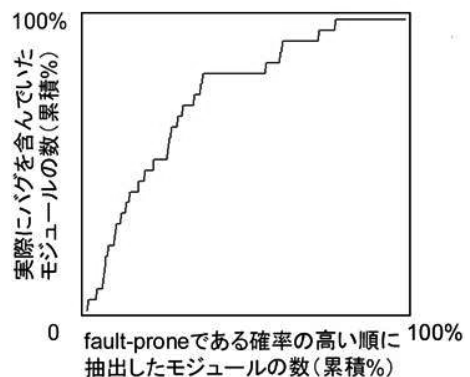
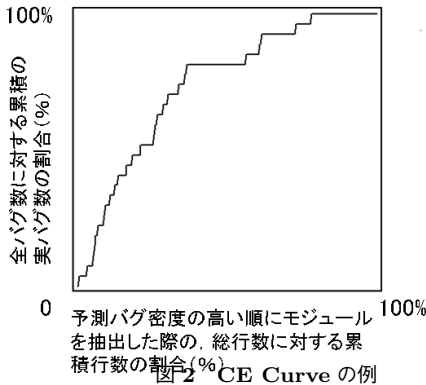


図1 Alberg Diagram の例



際の、総行数に対する累積行数の割合」、縦軸を「全バグ数に対する累積の実バグ数の割合」とした曲線の AUC として表される (本論文では、行数はコメント文を含むものとする)。現実のテストを想定した場合、含まれるバグの数が同じであればより規模の小さい (バグ密度の高い) モジュールをテストする方が少ない工数でバグを発見できることから、CE Curve の AUC はより現実に即した評価指標であるといえる。本論文では、バグモジュールの判別モデルの評価には Alberg Diagram の AUC を用いる事とし、バグ数予測、バグ密度予測の評価には CE Curve の AUC を用いる。

4 テスト戦略のシミュレーション

バグモジュール予測結果に基づいてテストを行った場合の発見バグ数を、シミュレーションにより推定する。

4.1 シミュレーション手順

以下にシミュレーションの手順を示す。

Step1. バグモジュール予測 : 過去の開発実績データから 3 つのバグモジュール予測モデル (ロジスティック回帰分析, ランダムフォレスト, 重回帰分析) を構築し, 予測対象データに適用することで予測値を得る。

Step2. テスト戦略に基づく工数割り当て : テスト戦略に基づいてテストモジュールに利用

可能な総テスト工数を分配する。テスト戦略の詳細は 4.2 節に記す。

Step3. 発見できるバグ数の期待値の導出 : Step2

で割り当てられたテスト工数とテストモジュールが与えられたときに、発見できるバグ数を求める。本論文では 4.3 節に示すバグ発見率モデルを用いて発見バグ数の導出を行う。

Step4. 発見バグ数の比較 : テスト戦略, 利用できるテスト工数をそれぞれ変化させて Step1 から Step3 を繰り返し, 発見できるバグ数の変化について分析する。

4.2 テスト戦略

本論文では、バグ予測の必要性を評価するために、バグ予測に基づかない代表的なテスト戦略である T0, T0' と、バグ予測に基づく戦略 T1~T4 を比較する。各戦略の詳細は次の通りである。

[テスト戦略 T0] $\text{テスト工数} \propto \text{モジュール規模}$: バグモジュール予測結果に基づかない戦略であり、各モジュールのテスト工数を、モジュールの規模に比例させて割り当てる。本戦略は、企業でも用いられている最も基本的なテスト戦略である [23]。

[テスト戦略 T0'] $\text{テスト工数} \propto \text{モジュールの複雑さ}$: バグモジュール予測結果に基づかない戦略であり、各モジュールのテスト工数を、モジュールの複雑さ (サイクロマティック数) に比例させて割り当てる。サイクロマティック数は単体テストにおける分岐網羅 (C1) を達成するのに必要なテストケース数の上限を表す。C1 カバレッジに基づくテストは多くの企業で用いられている。

[テスト戦略 T1] $\text{テスト工数} \propto \text{予測バグ数}$: バグ数の予測値に比例させたテスト工数を各モジュールに割り当てる事で、より多くのバグの発見を目指す。本戦略は、バグ予測に基づく最も基本的なテスト戦略である [14]。なお、バグの有無を予測した際には、その判別値 (バグが含まれる確率) に比例させてテスト工数を割り当てることとする。

[テスト戦略 T2] $\text{テスト工数} \propto \text{予測バグ密度}$: バ

グ密度の予測値に比例させたテスト工数を各モジュールに割り当てる事で、バグの発見しやすいものにテスト工数を多く割り当て、効率的なバグの発見を目指す。本戦略は、バグ密度が高い部分こそ重点的にテストすべきであるという考え方 [11] に基づいた戦略である。

[テスト戦略 T3] テスト工数 $\propto$ 予測バグ数 $\times$ モジュール規模: 本戦略 T3 と次の戦略 T4 は、戦略 T1 と T2 を組み合わせた戦略である。規模の大きいモジュールについてはより多くのテスト工数を割り当てないと全てのバグを発見しきれない可能性がある。そこで、本戦略では予測バグ数とモジュールの規模を掛け合わせた値に比例させたテスト工数を各モジュールに割り当てる事で、大規模かつ、バグの多く含まれているモジュールから確実にバグを発見する事を目指す。

[テスト戦略 T4] テスト工数 $\propto$ 予測バグ数 $\times \log(\text{モジュール規模})$: 一般に、ソフトウェアは少数の大規模モジュールと多数の小規模モジュールから構成されており、ソフトウェアの規模は正規分布に従っていないケースが多い事が知られている [5] [6]。単純に予測バグ数に規模を掛け合わせてしまうと、少数の大規模モジュールに過剰なテスト工数を割り当ててしまい、小規模モジュールのテスト工数が相対的に不十分となる可能性がある。そこで、モジュールの規模を対数変換し、予測バグ数に掛け合わせた値に比例させたテスト工数を各モジュールに割り当てる事で、少数の大規模モジュールをテストし過ぎないようにする。

4.3 バグ発見率モデル

バグモジュール予測の結果に基づいて 4.2 節のテスト戦略で各モジュールにテスト工数が割り当てられた場合、モジュールに含まれるバグはある確率で発見されることになる。本論文では、指数型 SRGM に着目し、モジュールに与えられたテスト工数で発見できるバグ数の推定を行う。指数型 SRGM はソフトウェアのテスト工程における残存バグ数の推定に用いられる [25]。指数型 SRGM は式 1 で表される。 $E(t)$ はバグ発見率を表し、この値に潜在バグ数を掛け合

わせたものが期待発見バグ数となる。 t は工数を表す。また、パラメータ b は単位工数当たりのバグ発見率であり、バグの発見容易性を表す。指数型 SRGM ではテスト工数 (時間) に応じて発見できるバグ数が増加するという事象を表している。

$$E(t) = 1 - \exp(-b \times t) \quad (1)$$

指数型 SRGM では以下の仮定に従ってソフトウェアのバグは発見される。

[仮定 1] テスト中に残存バグ数は増加しない

[仮定 2] バグの発見確率は時間とともに変化しない

[仮定 3] 全てのバグの発見容易性は等しい

従来、単位工数当たりのバグ発見確率を時間 (もしくは工数) の関数とする遅延 S 字型 SRGM なども提案されているが [28]、パラメータ推定が容易でないこと、および、現実のデータに対する適合度に大きな差がない事例もある [30] ことから、本論文では単位工数当たりのバグ発見確率を定数とする最も単純な指数型 SRGM を用いる。

ただし、モジュール毎に指数型 SRGM の適用を考えた場合、指数型 SRGM における上記の仮定 3 は成り立たない。なぜならば、モジュール毎で規模や複雑さが異なることから、仮に各モジュールに同様のテスト工数をかけたとしても、それぞれのバグ発見確率は異なると考えられるからである。そこで、本論文では指数型 SRGM を拡張し、モジュール毎のバグの発見容易性を考慮したバグ発見率モデルを提案する。モジュールにバグが含まれている場合、以下の仮定に従ってモジュール中のバグは発見されるとする。

[仮定 3'] バグの発見容易性はモジュールの複雑度 (サイクロマティック数 [15]) に反比例する。

モジュールの複雑度 S_i はモジュールの制御フローグラフの経路数であるサイクロマティック数を表す。サイクロマティック数は、テスト容易性の一尺度として従来からも利用されており [29]、単体テストにおいて完全な分岐網羅を達成するためのテストケース数の上限になる事が知られている。本仮定がどんなモジュールにも当てはまるとまでは言えないが、他の尺度よりも妥当性と計測容易性の観点から望ましいと考える。例えば、「ソースコード行数」は容易に計測可能

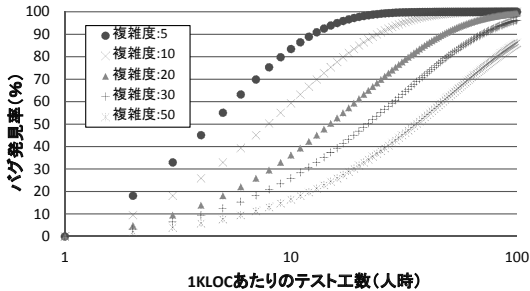


図3 バグ発見率モデルにおけるテスト工数、複雑度、バグ発見率の関係

であるが、制御フローの複雑さの違いが考慮されない点が問題である。また、条件網羅に対応する尺度として、分岐の条件式に含まれる AND や OR などの論理演算子の数を考慮した拡張サイクロマティック数も提案されている [21] が、多くの計測ツールでサポートされていない点が問題である。サイクロマティック数が大きいほど、分岐網羅を達成することが困難となり、バグ発見容易性が低下すると考えられる。そこで、本仮定では、バグ発見確率はサイクロマティック数に反比例するとしている。

提案するバグ発見率モデルは式 (1) の指数型 SRGM をベースとして式 (2) で表される。 $d(E_i)$ は、テスト工数 E_i をテスト予定モジュール $M_i (i=0,1,\dots,N)$ に割り当てた時のバグ発見確率を表す。 Z_i は、モジュール M_i のバグ発見容易性を表し、式 (3) で定義される。

$$d(E_i) = 1 - \exp(-Z_i \times E_i) \quad (2)$$

$$Z_i = \frac{Z_0}{S_i} \quad (3)$$

ここで、 Z_0 は定数、 S_i はモジュール i の複雑度 (サイクロマティック数) を表す。

式 (2) に基づいてバグ発見数を推定する場合、モジュール M_i の含有バグ数を Br_i とすると、バグの期待発見数 Be_i は式 (4) で与えられる。

$$Be_i = Br_i \times d(E_i) \quad (4)$$

このモデルでは、モジュール中のバグの発見確率はテスト工数に比例し、モジュールの複雑度 (サイクロマティック数) に反比例するという事象を表している。モジュールにただ 1 つのバグが潜在する場合、図 3 に示すようにテスト工数をかけるほどバグ発見率が

上がり、モジュールの複雑度が高ければバグ発見により多くのテスト工数を要する。

5 シミュレーション実験

1 章で述べたりサーチクエスチョンに答えるために、バグモジュール予測の結果に基づきテスト工数をモジュールに割り当てた時、テスト戦略、利用できるテスト工数がバグ発見率にどう影響を及ぼすかについてバグ発見率モデルを用いたシミュレーションを行う。

5.1 パラメータ設定

バグ発見率モデルに用いるパラメータは、ソフトウェア開発データ白書 [8] に記載されている開発実績データに基づいて次のように決定した。

- 総テスト工数 E_{ALL} : 1KLOC 当たり 32.7 (人時)
- バグの発見容易性のパラメータ Z_0 : 8.72

これらの値の算出にあたっては、1 つの (平均的な) プロジェクトを 1 つのモジュールと捉え、白書におけるエンタープライズ系システム (企業の業務システム) の平均のサイクロマティック数、総テスト工数 (E_{ALL})、テストでの検出バグ数、テストでの見逃しバグ数 (出荷後 6ヶ月間の発見バグ数) を式 (2),(3) にあてはめて Z_0 を推定することとした。ただし、白書にはサイクロマティック数は掲載されていない。次善策として、Eclipse のデータから導出した「行数からサイクロマティック数の換算式」に白書記載の平均ソースコード行数を代入し、平均サイクロマティック数を推定し、 Z_0 の推定に用いることとした。また、 Z_0 は、出荷後 6ヶ月に発見されたバグを、開発中のテストで発見できなかったバグとみなすことで、算出した。本論文では、総テスト工数 E_{ALL} を標準的なテスト工数として扱い、RQ1 のシミュレーションにおけるテスト工数として用いる。

5.2 データセット

実験にはオープンソースソフトウェア (OSS) の開発データを用いた。本実験では OSS の 2 つのプロジェクト (Eclipse, Mylyn) を実験対象として、バグモジュール予測を行った。各プロジェクトにおいて、古いバージョンのデータ (モジュール特性とバグ数)

表 1 データセット概要 (Eclipse)

	v3.0	v 3.1	v 3.2
モジュール数	4,629	5,462	6,512
総行数 (KSLOC)	611	738	883
総バグ数	3,030	3,191	2,571

表 2 データセット概要 (Mylyn)

	v1.0	v 2.0	v 3.0
モジュール数	1,009	1,231	1,503
総行数 (KSLOC)	157	192	241
総バグ数	1,611	2,611	1,041

表 3 メトリクスの概要

メトリクス	概要	メトリクス	概要
TLOC	総行数	NOM	メソッド数
FOUT	メソッドの呼び出し数	NSF	静的フィールド数
MLOC	論理行数	NSM	静的メソッド数
NBD	ネスト深さ	ACD	匿名型宣言数
PAR	パラメーター数	NOI	インターフェース数
VG	サイクロマティック数	NOT	クラス数
NOF	フィールド数		

表 4 バグモジュール予測方法

分類	目的変数	値域	モデル化手法
バグモジュール判別	バグの有無 (判別値)	[0,1]	LR,RF
バグ密度予測	バグ密度	[0,1]	LM,RF
バグ数予測	バグの数	[0, ∞]	LM,RF

をフィットデータとしてバグモジュール予測モデルを構築し、新しいバージョンの各モジュールに対して予測を行った。

表 1 に Eclipse データセットの概要を示し、表 2 に Mylyn データセットの概要を示す。本論文では 13 種類のメトリクスを収集し、予測モデルの説明変数として用いた (表 3)。

本実験では、表 4 の 3 種類のバグモジュール予測手法を用いた。バグモジュール判別モデルの構築にはロジスティック回帰分析 (LR)、ランダムフォレスト法 (RF) を用いた。バグ数予測、バグ密度予測モデルの構築には重回帰分析 (LM)、ランダムフォレスト法 (RF) を用いた。

6 結果

6.1 各モデルの予測精度

表 5 と表 6 に本実験で構築したバグモジュール予測モデルの各目的変数ごとの AUC を載せる。表中の「モジュール」の列は、モデル構築に用いたバー

表 5 AUC(Eclipse)

		目的変数					
		バグの有無		バグ数		バグ密度	
モジュール		LR	RF	LM	RF	LM	RF
v3.0	v3.1	0.73	0.72	0.71	0.72	0.73	0.69
v3.0	v3.2	0.76	0.76	0.70	0.72	0.71	0.70
v3.1	v3.2	0.74	0.75	0.66	0.70	0.71	0.69

表 6 AUC(Mylyn)

		目的変数					
		バグの有無		バグ数		バグ密度	
モジュール		LR	RF	LM	RF	LM	RF
v1.0	v2.0	0.57	0.55	0.68	0.69	0.67	0.67
v1.0	v3.0	0.67	0.68	0.72	0.73	0.71	0.70
v2.0	v3.0	0.71	0.71	0.72	0.74	0.72	0.72

ンと予測を行ったバージョンの組合せを表している。各行はそれぞれのバージョン間予測における AUC の値を表している。

6.2 RQ1：最も効果の高いテスト戦略は何か

RQ1 に対する実験結果を表 7, 8, 9, 10 に示す。表 7, 8 中の値は各バージョン間予測の組み合わせでの最もバグ発見率の高いテスト戦略を表している。これらの表中「なし」の部分は表 5, 表 6 において AUC の値が 0.7 未満 (すなわちバグモジュール予測精度が低い) のためにテスト工数割り当てを行わなかった事を示す。表 9, 10 は、各テスト戦略におけるバグ発見率の平均値を示す。

表 7 より、Eclipse プロジェクトにおいては評価対象の 15 個のテスト工数割り当て中、8 つのテスト工数割り当てで、T4: テスト工数 $\propto$ 予測バグ数 $\times \log$ (モジュール規模) が最もバグ発見率の高いテスト戦略になっている。また、表 9 より、バグ発見率の平均値においても、テスト戦略 T4 が最も高いバグ発見率を示した (平均バグ発見率 76 %)。Eclipse プロジェクトに対するテスト工数割り当てでは、テスト戦略 T4 が最も総バグ発見率の高いテスト戦略だといえる。

同様に、表 8 より、Mylyn プロジェクトにおいてもテスト戦略 T4 が最もバグ発見率の高いテスト戦略であった。評価対象の 10 個のテスト工数割り当て中、8 つのテスト工数割り当てで、T4: テスト工数 $\propto$

表 7 バグ発見率の高いテスト戦略 (Eclipse)

		目的変数					
		バグの有無		バグ数		バグ密度	
モジュール		LR	RF	LM	RF	LM	RF
v3.0 v3.1	T3	T4	T0	T1	T1	なし	
v3.0 v3.2	T4	T3	T4	T4	T1	T4	
v3.1 v3.2	T3	T3	なし	T4	T1	なし	

表 8 バグ発見率の高いテスト戦略 (Mylyn)

		目的変数					
		バグの有無		バグ数		バグ密度	
モジュール		LR	RF	LM	RF	LM	RF
v1.0 v2.0	なし	なし	なし	なし	なし	なし	なし
v1.0 v3.0	なし	なし	T4	T4	T4	T4	
v2.0 v3.0	T1	T1	T4	T4	T4	T4	

表 9 バグ発見率の平均値 (Eclipse)

		目的変数	
テスト戦略	バグの有無	バグ数	バグ密度
T0		73 %	
T0'		73 %	
T1	73 %	74 %	77 %
T2		55 %	62 %
T3	75 %	70 %	58 %
T4	75 %	76 %	77 %

表 10 バグ発見率の平均値 (Mylyn)

		目的変数	
テスト戦略	バグの有無	バグ数	バグ密度
T0		84 %	
T0'		84 %	
T1	79 %	85 %	85 %
T2		69 %	71 %
T3	87 %	80 %	70 %
T4	82 %	87 %	86 %

予測バグ数 $\times \log(\text{モジュール規模})$ 戦略が最も総バグ発見率の高いテスト戦略となっている。また、表 10 より、総バグ発見率の平均値においても、テスト戦略 T4 が最も高いバグ発見率を示した (平均バグ発見率 85 %)。

以上の結果より、RQ1 の回答としては、テスト工数を“予測バグ数 $\times \log(\text{モジュールの規模})$ ”に比例させてモジュールに割り当てる戦略が最有力であるといえる。

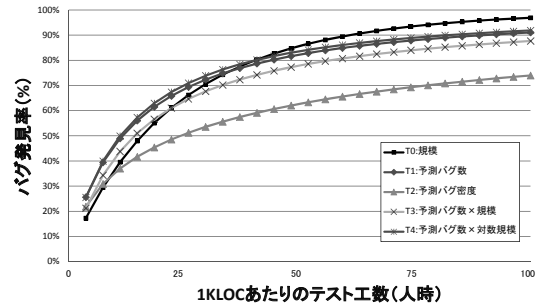


図 4 利用できるテスト工数とバグ発見率の関係 (Eclipse)

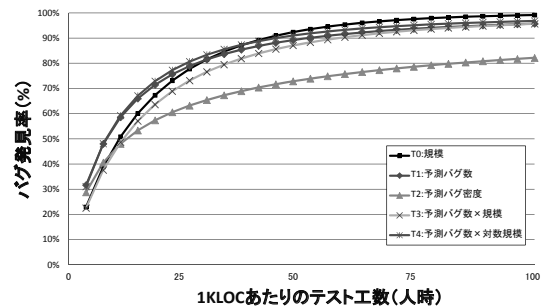


図 5 利用できるテスト工数とバグ発見率の関係 (Mylyn)

6.3 RQ2：総テスト工数は有効なテスト戦略にどのような影響を与えるか

本節では、テスト工数を変化させて各テスト戦略の下でのバグ発見率の変化について分析する。

実験結果を図 4 と図 5 に示す (戦略 T0' は T0 とほぼ同じ結果であったため省略している)。この図は、各戦略について、3 つの目的変数、3 つの予測モデル (LR, LM, RF)、および、3 つのバージョン間予測の結果を平均した結果を表している。図 4 の縦軸はバグ発見率を表し、横軸は 1KLOC 当たりの平均テスト工数を表す。図中のプロットは各テスト戦略を表す。図 5 においても同様である。

図 4 と図 5 より Eclipse と Mylyn の両方の結果で、RQ1 における標準的なテスト工数 (32.7 人時/KLOC) の範囲内では、テスト戦略 T4：テスト工数 $\propto$ 予測バグ数 $\times \log(\text{モジュール規模})$ 、T1：テスト工数 $\propto$ 予測バグ数が最もバグを発見できるテスト戦略である事がみてとれる。また、利用できるテスト工数を増加させ

ていくと、テスト戦略 T0:テスト工数 $\propto$ モジュール規模が最もバグの発見できるテスト戦略となっている。

利用できるテスト工数を増加させるにつれて、テスト戦略 T0 のバグ発見率がテスト戦略 T1,T4 のバグ発見率を上回った。この理由として、バグモジュール予測における予測誤りの影響が考えられる。バグモジュール予測を用いるテスト戦略では、予測誤りによりバグが含まれるにも関わらずテスト工数をほとんど割り当てられないモジュールが存在する。そのため、バグモジュール予測結果を用いたテスト戦略ではテスト工数を増加させた時のモジュール全体でのバグ発見率がバグモジュール予測を用いないテスト戦略に比べて低下する。

上記の実験結果より、RQ2 に対する回答としては、標準的なテスト工数の範囲では、バグモジュール予測を用いたテスト戦略 (T1,T4) が最も多くのバグを発見できるが、利用できるテスト工数が増えると、バグモジュール予測を用いないテスト戦略 T0(もしくは T0') が最も多くのバグを発見できるといえる。この結果は、多大なテスト工数が費やされるような高信頼性システムでは、バグモジュール予測を行わず、各モジュールの規模 (または複雑さ) に応じたテストが望ましいことを示唆している。

7 まとめ

本論文では、バグモジュール予測モデルの予測結果に基づいたモジュールへのテスト工数の割り当て方法 (テスト戦略) について、シミュレーション実験により評価を行った。本論文で得られた主な知見は以下のとおりである。

- テスト工数を“モジュールの予測バグ数 $\times \log$ (モジュール規模)”に比例させて割り当てるテスト戦略が最も多くのバグを発見できる。
- 標準的なテスト工数の下では、バグモジュール予測結果に基づいてテストをすることでより多くのバグの発見が期待できる。
- 利用できるテスト工数が増えると、バグモジュール予測を用いないテスト戦略が最も多くのバグを発見できる。

開発現場ではこれらの知見を踏まえてバグモジュール

予測モデルを利用する事でよりテスト工程での利用に即したバグモジュール予測を行う事が期待できる。今後の課題としては、より多くのテスト戦略について、評価を行うことが重要となる。また、バグ発見率モデルのパラメータ Z_0 はプロジェクトによって異なると考えられるため、 Z_0 を変化させたときにテスト戦略の優劣が変わり得るかについての評価が必要となる。また、バグ発見率モデルを現実のデータに当てはめ、モデル自体の妥当性を定量的に評価することも重要な課題である。さらに、実際に本論文で提案したテスト戦略を実証的に評価し、実際の開発現場でのバグモジュール予測を用いたテスト工数割り当ての利用効果について明らかにする事である。

参考文献

- [1] 阿萬裕久: 論理的制約条件付き 0-1 計画モデルを用いた重点レビュー計画法, コンピュータソフトウェア, Vol. 29, No. 3 (2012), pp. 115–120.
- [2] D'Ambros, M., Lanza, M. and Robbes, R.: An extensive comparison of bug prediction approaches, in *Proc. Working Conf. on Mining Software Repositories*, 2010, pp. 31–41.
- [3] Arisholm, E., Briand, L.C. and Fuglerud, M.: Data mining techniques for building fault-proneness models in telecom java software, in *Proc. 18th IEEE International Symposium on Software Reliability Engineering*, 2007, pp.215–224.
- [4] Breiman, L.: Random Forests, *Machine Learning*, Vol. 45, No.1(2001), pp. 5–32.
- [5] 古山垣夫: ソフトウェアプロジェクトデータの量的変数に関する分析の一指針と分析事例, SEC journal Vol. 7, No. 3 (2011), pp. 105–111.
- [6] Glass, L.: *Facts and fallacies of software engineering*, Addison-Wesley Professional, 2003.
- [7] 林卓磨, 信田祥司, 白井久美子, 吉田順一, 門田曉人, 松本健一: Fault-prone モジュール判別モデルを用いた検証項目数削減に関する一検討, 2011 年電子情報通信学会ソサイエティ大会通信講演論文集 2, B-18-2, 2011, p. 406.
- [8] 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 2010–2011, 情報処理推進機構, 2010.
- [9] 亀井靖高: ソフトウェアモジュールの信頼性予測の精度向上に関する研究, 奈良先端科学技術大学院大学博士論文, NAIST-IS-DD0761011, 2009.
- [10] 柿元健, 門田曉人, 靖高亀井, 松本真佑, 松本健一, 楠本真二: 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, 情報処理学会論文誌, Vol. 50, No. 7 (2009), pp. 1716–1724.
- [11] Knab, P., Pinzger, M. and Bernstein, A.: Predicting Defect Densities in Source Code Files with De-

- cision Tree Learners, in *Proc. 3rd Working Conference on Mining Software Repositories*, 2006, pp. 119–125.
- [12] 額部伸子, 川村真弥, 野村准一, 野中誠: プロセスおよびプロダクトメトリクスを用いた Fault-Prone クラス予測の適用事例, 情報処理学会研究報告, ソフトウェア工学研究会, Vol. 2010-SE-168, No. 6 (2010), pp. 1–8.
- [13] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Transactions on Software Engineering*, Vol. 34, No. 4(2008), pp. 485–496.
- [14] Li, P., Herbsleb, J., Shaw, M. and Robinson, B.: Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc., in *Proc. 28th International Conference on Software Engineering*, 2006, pp. 413–422.
- [15] McCabe, T.: A complexity measure, *IEEE Transactions on Software Engineering*, No. 4 (1976), pp. 308–320.
- [16] Mende, T. and Koschke, R.: Revisiting the evaluation of defect prediction models, in *Proc. 5th International Conference on Predictor Models in Software Engineering*, 2009, pp. 1–7.
- [17] Menzies, T., Greenwald, J. and Frank, A.: Data mining static code attributes to learn defect predictors, *IEEE Transactions on Software Engineering*, Vol. 33, No. 1 (2007), pp. 2–13.
- [18] Monden, A., Hayashi, T., Shinoda, S., Shirai, K., Yoshida, J., Barker, M. and Matsumoto, K.: Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing, *IEEE Transactions on Software Engineering*, preprint available, DOI Bookmark: <http://doi.ieeecomputersociety.org/10.1109/TSE.2013.21>
- [19] 門田暁人, 小林健一: 線形重回帰モデルを用いたソフトウェア開発工数予測における対数変換の効果, コンピュータソフトウェア, Vol. 27, No. 4 (2010), pp. 234–239.
- [20] Myers, G.J.: *The Art of Software Testing*, Wiley, New York, 1979.
- [21] Myers, G.J.: An extension to the cyclomatic measure of program complexity, *ACM SIGPLAN Notices*, Vol. 12, No. 10 (1977), pp. 61–64.
- [22] Nagappan, N., Ball, T. and Zeller, A.: Mining metrics to predict component failures, in *Proc. 28th International Conference on Software Engineering*, 2006, pp. 452–461.
- [23] 奈良隆正: ソフトウェア品質検証, 評価技術の勘所, JaSST'09 Kansai, 2009.
- [24] Ohlsson, N. and Alberg, H.: Predicting fault-prone software modules in telephone switches, *IEEE Transactions on Software Engineering*, Vol. 22, No. 12 (1996), pp. 886–894.
- [25] Ramamoorthy, C.V. and Bastani, F.: Software reliability: status and perspectives, *IEEE Transactions on Software Engineering*, No. 4 (1982), pp. 354–371.
- [26] 佐藤慎一, 端山毅, 門田暁人, 松本健一: ソフトウェア複雑度メトリクスに基づく大規模システムの欠陥モジュール分類とその評価, 第 19 回ソフトウェア生産における品質管理シンポジウム報告集, 1999.
- [27] Turhan, B., Menzies, T., Bener, A. and DiStefano, J.: On the relative value of cross-company and within-company data for defect prediction, *Empirical Software Engineering*, Vol. 14, No. 5 (2009), pp. 540–578.
- [28] Yamada, S., Ohba, M. and Osaki, S.: S-shaped reliability growth modeling for software error detection, *IEEE Transactions on Reliability*, Vol. 32, No. 5 (1983), pp. 475–484.
- [29] 山田茂, 高橋宗雄: ソフトウェアマネジメントモデル入門, 共立出版, 1993.
- [30] Yamada, S. and Osaki, S.: Software reliability growth modeling: models and applications, *IEEE Transactions on Software Engineering*, Vol. SE-II, No. 12 (1985), pp. 1431–1437.



中野 大輔

2009 年奈良工業高等専門学校専攻科電子情報工学専攻科修了。2011 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、株式会社日立製作所勤務。



門田 暁人

1994 年名古屋大学工学部電気学科卒業。1998 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。同年同大学同研究科助手。2004 年同大学助教授。2007 年同大学准教授。2003~2004 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等の研究に従事。日本ソフトウェア科学会, 電子情報通信学会, 情報処理学会, IEEE, ACM 各会員。



亀井 靖高

2005 年関西大学総合情報学部卒業。2007 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。2009 年同大学院博士後期課程修了。同年日本学術振興会・特別研究員 (PD)。2010 年カナダ・Queen's 大学・博士研究員。2011 年九州大学大

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等の研究に従事。IEEE, IEICE, IPSJ 各会員。



松本 健一

1985 年大阪大学基礎工学部情報工学科卒業。1989 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。1993 年奈良先端科学技術大学院大学助教授。2001 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。日本ソフトウェア科学会, 電子情報通信学会, 情報処理学会, ACM 各会員, IEEE Senior Member.