バグモジュール予測を用いたテスト工数割り当て戦略

Effort Allocation Strategies in Software Testing Using Bug Module Prediction

中野 大輔[1]　門田 暁人[2]　松本 健一[3]

あらまし　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きた．しかし，予測結果に基づくテスト方法（テスト戦略）は従来，ほとんど研究されていない．本論文では，テストの効果を評価するシミュレーション方法を提案し，Eclipseデータセットを用いて多数のテスト戦略を比較する．シミュレーションの結果，テスト工数を“モジュールの予測バグ数×log（モジュール規模）”に比例させる戦略が最も効果が高いことが分かった．

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，ソフトウェア品質保証活動（インスペクション，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．

ソフトウェアインスペクションやテスト工数の効率的な配分を行うために，バグモジュール予測方法が盛んに研究されてきた．それらの研究では，ソフトウェアに含まれる各モジュールの(a)バグの有無[9]，(b)バグ数[1][4]，(c)バグ密度[6]などの予測が行われる．予測の元となるのは，多数のモジュールメトリクス（規模やサイクロマティック数などのプロダクトメトリクス，および，修正回数などのプロセスメトリクス）であり，線形回帰モデル，ロジスティック回帰モデル，ニューラルネット，決定木，ランダムフォレストなど，多種多様な予測モデルが用いられてきた[3]．開発者は，予測結果に基づき，限られたテスト（インスペクション）リソースをバグモジュール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．

しかし，バグモジュール予測の結果に基づいたテスト戦略については(柿元の研究[10]を除いて）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確保の効果は明らかにされていない．たとえバグモジュールの予測精度が高くても，テスト戦略が悪ければ，テスト工数をかえって増大させたり，品質を低下させる可能性がある．そこで，本論文では，次の２つのリサーチクエスチョンを明らかにすることを目的とする．

（RQ1）テスト工数割り当てに有効なバグモジュール予測の目的変数は何か：(a)バグの

[1] Daisuke Nakano,　奈良先端科学技術大学院大学
[2] Akito Monden,　奈良先端科学技術大学院大学
[3] Ken-ichi Matsumoto,　奈良先端科学技術大学院大学

有無，(b)バグ数，(c)バグ密度のいずれであるか.

バグモジュール予測に基づくテスト戦略を考えるにあたって，そもそもバグモジュールの予測において何を目的変数として予測すべきかについて，従来さまざまな立場があり，明確なコンセンサスが得られていない．多くの従来研究では(a)が採用され，バグを1個以上含むモジュール（fault-prone module）と含まないモジュール（not fault-prone module）を区別することが目的とされている．柿元[10]のテスト戦略においても(a)を採用している．この場合，バグの数については考慮されない．その理由として，2 個以上のバグを含むモジュールは数が多くないことから，バグの有無を判別するだけで十分であるという判断がある[9]．一方，実務者は，(b)を採用し，予測バグ数に応じてテスト工数を割り当てる場合がある[4]．また，いくつかの研究では，(c)を採用している[1]．その理由は，規模が大きいモジュールほどバグを含みやすいのは当然であるので，バグの有無を予測しても有益な情報とはいえず，高いバグ密度を持つモジュールを予測した方が役に立つという考え方に立つものである．(a)(b)(c)のいずれが最適であるかは，テスト工数割り当て戦略に依存するため，次の RQ2 に答えることが必要となる．

（RQ2）バグモジュール予測に基づく有効なテスト戦略は何か.

テスト戦略としては様々なものが考えられる．柿元[10]は，バグの有無の予測結果に基づいて，バグありと予測されたモジュールに対し，バグなしと予測されたモジュールの n 倍の工数を割り当てる戦略を提案し，最適な n の値についてシミュレーションによる評価を行っている．ただし，柿元の研究では，モジュールの規模や複雑さを考慮していない．一般に，複雑なモジュールほどバグ発見が困難であることから，モジュールの複雑さを考慮した評価が必要となる．また，各モジュールのテスト工数を，バグ数の予測値に比例させて割り当てる戦略も知られているが[1]，その有効性については従来評価されていない．他に，バグ密度に比例させてテスト工数を配分することも考えられる．どのテスト工数が有効であるかは，利用可能な総テスト工数にも影響されると考えられる．バグ予測には必ず予測誤りが含まれることから，テスト工数に余裕がある場合，全てのモジュールをまんべんなく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．本論文では，いずれのテスト戦略が有効であるかについて，利用可能な総テスト工数との関係を明らかにする．

本論文では，RQ1 と RQ2 に答えるために，バグ予測結果に基づいたテスト工数割り当て方法（テスト戦略）を提案するとともに，割り当てた工数と発見できるバグ数の期待値のモデル化（バグ発見率モデル）を行う．バグ発見率モデルを用いたシミュレーション実験を通じて，各テスト戦略での発見バグ数の変化を分析し，有効性を明らかにする．

2　関連研究

従来，バグモジュール予測の研究が盛んに行われており，数多くのモデル化手法や予測精度向上への取り組みが報告されている．また予測精度向上のみならず，企業でのバグ予測の適用事例も数多く報告されている．文献[4]では企業でのバグモジュール予測の適用効果を定性的に示した事例が報告されており，文献[7]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．

Mende[6]はモジュールのバグ密度（単位行数当たりのバグ数）を目的変数として用いてバグモジュール予測を行っており，予測モデルの評価指標としてモジュールの規模を考慮した P_{opt} を提案した．バグ密度の予測を行い，P_{opt} を用いる事でテスト工程でのバグの発見効率が高いモデルを評価できるとしている．しかしながら，いずれの研究においてもモデル化の手法や予測精度の向上にとどまっており，予測後の具体的なアクションについては明らかではない．

予測結果に基づいたアクションについて述べた文献として，柿元[10]は，バグの有無の予測結果に基づいたテスト工数割り当てのシミュレーションを行い，割り当てたテスト工数で発見できるバグの期待数をモデル化し，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た．しかし，モジュールごとにバグは一様な確率で発見されるという仮定でモデル化を行っている点は現実的とはいえない．またバグ数やバグ密度の予測値に基づくテスト工数戦略についても検討されていない．

3　バグモジュール予測を用いたテスト工数割り当て

3.1　バグモジュール予測

本論文ではモデル化手法にロジスティック回帰分析，（線形）重回帰分析，ランダムフォレストの 3 種類のモデル化手法を用いた．それぞれのモデル化手法は現在までに広く用いられている．ロジスティック回帰分析はモジュールにバグが含まれるか否かという 2 値判別モデルに最も利用されており，本論文でもバグモジュール判別にロジスティック回帰分析を用いる．重回帰分析は目的変数がバグの数や密度といった連続値の場合に用いられ，多くの研究で用いられている．ランダムフォレストは分類木，または回帰木を用いて集団学習を行うモデル化手法であり，モデル構築データに過度に適合（オーバーフィッティング）しにくい特徴を持ち，目的変数が 2 値，連続値どちらの場合でも用いる事ができる．ランダムフォレストは，今日，もっとも有望なモデルの一つである[3]．

本論文では，バグモジュール判別を行う際のモデル化手法にはロジスティック回帰分析，ランダムフォレストを用いる．バグ数予測，バグ密度予測を行う際のモデル化手法には重回帰分析，ランダムフォレストを用いる．表 1 に，本論文で用いるモデル化手法の概要を示す．表中，LR はロジスティック回帰分析，LM は重回帰分析，RF はランダムフォレストを示す．

表 1　モデル化手法の概要

分類	目的変数	値域	モデル化手法
バグモジュール判別	バグが含まれているか否か	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

バグモジュール予測の結果に基づきテスト工数を割り当てる際に，テスト戦略，総テ

スト工数だけではなく，モデル自体の予測精度も考慮に入れるべき要素である．本論文では予測モデルの評価指標として Alberg Diagram[8]の AUC と Mende[6]の提案した P_{opt} を用いる．Alberg Diagram とはバグを含んでいる可能性の高い順番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表し，縦軸は実際にバグを含んでいたモジュールの数を表す．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，グラフの曲線下面積の値は上昇する．AUC の値域は[0,1]であり，一般的に AUC の値が 0.8 を超えると予測精度が高い事が知られている．また，ランダムに予測を行った場合 AUC はおおよそ 0.5 となる．

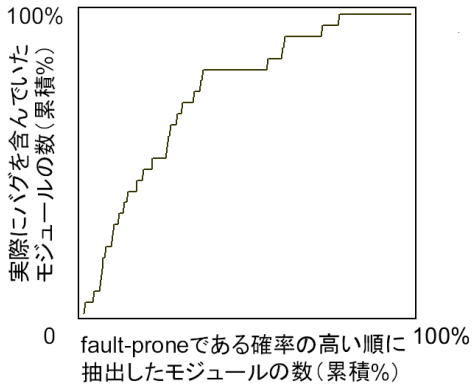

図 1　AUC の概要

AUC の値が高いほど，少ないモジュール数を調べる事でモジュールに含まれるバグをより多く発見できることになる．AUC は従来用いられてきた F1 値や再現率，適合率などを用いた評価方法と異なり，連続値で得られる結果を 2 値判別の結果に区切るしきい値に依存しない事や，単一評価が可能，という利点がある．

対して，P_{opt} 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．テストを行う事を想定した場合，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）モジュールを予測するモデルが望ましい．AUC では規模の大小について考慮しないが，P_{opt} では横軸にモジュールの累積行数を取ることで，バグ密度が大きいモジュール（規模が小さいわりにバグ数の多いモジュール）を誤判別するモデルに対して低い評価を行う．P_{opt} の値域は AUC と同じく[0,1]であり，値が 1 に近い程予測精度が高い事を示す．本論文では，バグモジュール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指標には P_{opt} を用いる．

柿元[10]はバグモジュール判別モデルにおいて予測精度とバグ発見率の関係について述べており，ある程度の予測精度が得られなければ予測結果に基づくテスト工数割り当ての効果が得られないとしている．そこで，本論文では AUC，P_{opt} の値が 0.7 以上の予測モデルを用いて予測値に基づくテスト工数割り当てを行いバグ発見率のシミュレーションを行う．

3.2　テスト工数割り当て戦略

各モジュールに対するテスト工数割り当て戦略は様々なものが考えられる．本論文では，次の 5 つの戦略 T0～T4 の有効性を評価する．

- **戦略 T0： テスト工数∝モジュール規模**
  本戦略は，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，企業において用いられている最も基本的な戦略である．
- **戦略 T1： テスト工数∝予測バグ数**
  バグモジュール予測の結果，予測されるバグの数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．なお，バグの有無を予測した場合は，バグありの確率に比例させてテスト工数を割り当てる．
- **戦略 T2： テスト工数∝予測バグ密度**
  バグ密度の高いモジュールは少ないテスト工数で多くのバグを発見できる可能性が高い．バグモジュール予測の結果，予測されるバグ密度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られることになり，効率的なバグの発見を目指す．
- **戦略 T3： テスト工数∝予測バグ数×モジュール規模**
  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，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，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しようとする戦略である．
- **戦略 T4： テスト工数∝予測バグ数×log（モジュール規模）**
  一般的に，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いる事が多い．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．

上記のテスト戦略の評価を，後述するバグ発見率モデルを用いて行う．

3.3 バグ発見率モデル

本論文では，テスト予定モジュールに対して，3.2 節の各テスト戦略に基づいてテスト工数を割り当てた場合のバグの発見率を求める．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．

・テスト工数に比例してバグの発見率は増加する

・モジュールの複雑度（サイクロマティック数[5]）に反比例してバグの発見率は低下する

あるテスト戦略 T_0 に基づき総テスト工数 E_{ALL} の内，テスト工数 E_i をテスト予定モジュール M_i（i=0,1,…N）に対して割り当てたとする．モジュール M_i の含有バグ Br_i はある確率 $d(E_i)$ で発見されるとし，割り当てたテスト工数 E_i で発見できるバグの期待発見数 Be_i は式（1.1）で求められる．

$$Be_i = Br_i \times d(E_i) \qquad (1.1)$$

本論文では式(1.1)をバグ発見率モデルとする．確率 $d(E_i)$ は式(1.2)で表される．また，バグ発見率モデル中の Z_i はモジュール毎のバグ発見効率を表し，式(1.3)で定義される．

$$d\ (E_i) = 1 - \exp(-Z_i E_i) \qquad (1.2)$$

$$Z_i = \frac{Z_0}{S_i} \qquad (1.3)$$

バグ発見率モデルは指数関数モデルを用いて表現される．指数関数モデル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ソフトウェア信頼度成長モデル[12]にも使用されている．ソフトウェア信頼度成長モデルでは時間経過後の発見可能バグ数の見積もりを行うのに対して，バグ発見率モデルではあるテスト工数を割り当てた時の期待バグ発見数を推定する．一般的にモジュール中のバグはテスト工数をかければかけるほど発見できる確率が上昇すると考えられるが，モジュール毎でバグの発見確率が異なる事も考慮しなければならない．バグ発見率モデルにおいて，モジュール毎のバグの発見効率は Z_i で表される．式(1.3) より，Z_i は Z_0 と S_i の商で求められ，Z_0 は単位工数当たりのバグの発見率，S_i はモジュールの複雑度（サイクロマティック数）を表す．単位工数当たりのバグの発見率 Z_0 は定数であり，モジュールのバグ発見確率は実質的にはモジュールの複雑度 S_i に反比例する事になる．モジュールの複雑度 S_i はソースコードの制御フローグラフの経路数を表し，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．単位工数当たり当りのバグ発見率 Z_0 は定数としたが，ソフトウェア信頼度成長モデルによっては Z_0 を時間（もしくは工数）の関数とする場合もある．しかし，パラメータ推定が容易でないことから，本論文ではもっとも簡単な指数関数モデルを踏襲し，Z_0 を定数としている．

本論文で用いるバグ発見率モデルは，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．バグ発見率モデルのもとでは，モジュールのただ 1 つのバグが潜在する場合，図 2 に示すようにテスト工数をかけるほどモジュール中のバグは発見できる事になり，モジュールの複雑度が高ければモジュール中のバグはより多くのテスト工数をかけなければ発見できない．式(1.1)，(1.2)より，あるテスト戦略 T に基づき各モジュールに割り当てられたテスト工数 E_i で発見できるバグの発見率 B_T は期待発見バグ数と含有バグ数の商の総和で求められ，式(1.4) で表す事ができる．本論文では，バグ発見率モデルを用いて 3.2 節の各テスト戦略(T0，T1，T2，T3，T4) で発見できる総バグ数について式(1.4) を用いてシミュレーションを行い，テスト戦略の評価を行う．

$$B_T = \sum_{i=0}^{N} \frac{Be_i}{Br_i} \qquad (1.4)$$

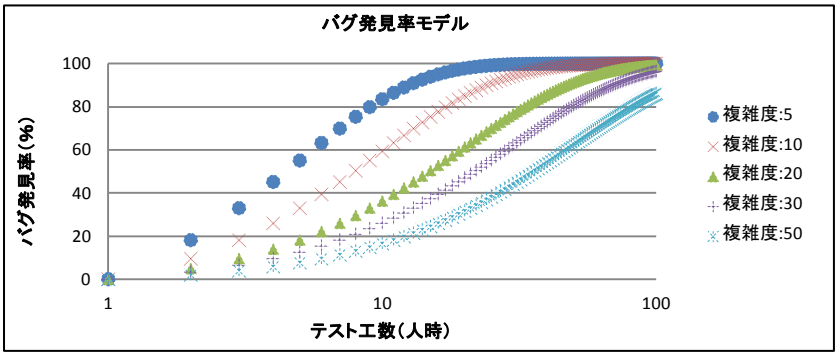

図 2　バグ発見率モデルにおけるテスト工数，複雑度，バグ発見率の関係

4　シミュレーション実験

4.1　実験設定

本論文のシミュレーション実験は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，(1)バグモジュール予測の目的変数(2)テスト戦略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て明らかにする．

バグ発見率モデルに用いるパラメーター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公開されているソフトウェア開発企業のテスト工程の開発データ[11]を参考に以下のように値を決定した．

- **総テスト工数：** E_{ALL} ＝ 1 モジュールにつき 4.4（人時）
- **単位工数当たりのバグ発見率：** Z_0 ＝ 8.72

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る．実験では Eclipse プロジェクトが開発する Eclipse　Platform の複数のバージョン（ v3.0，v3.1，v3.2）を対象として，バージョン間の予測を行った．バージョン間の予測では，v3.0 のデータセットから v3.1 と v3.2 の予測，v3.1 のデータセットから v3.2 の予測を行った．データセットの概要を表 2 に示す．

表 2　データセット概要

バージョン	v3.0	v3.1	v3.2
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

また，それぞれのモジュールから収集した静的メトリクスの概要を表 3 に示す．本論文では 13 種類のメトリクスを収集した．

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	ネスト数	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	複雑度	NOT	クラス数
NOF	フィールド数		

4.2 実験結果

4.2.1 モデルの評価指標

表 4 に本実験で構築したバグモジュール予測モデルのそれぞれの目的変数ごとの AUC と P_{opt} を載せる. 各行はそれぞれのモジュール間予測における AUC と P_{opt} の値を示している．3.1 でも述べたように，AUC もしくは P_{opt} の値が 0.7 以下のモデルの予測結果に対してはテスト工数割り当てを行わないものとする.

表 4 AUC と P_{opt}

		目的変数					
		バグの有無		バグ数		バグ密度	
バージョン		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

（LR:ロジスティック回帰　LM:重回帰　RF:ランダムフォレスト）

4.2.2 RQ1 に対する実験結果

RQ1 に対する実験結果を表 5 に示す. 各行は 3.3 のテスト戦略での各目的変数の総バグ発見率を示している. 表中の値はモジュール間予測（3.0→3.1, 3.0→3.2, 3.1→3.2）での総バグ発見率の平均値を示す．バグの有無を目的変数としてモデルを構築した場合は予測バグ密度を求める事ができないため，テスト戦略 T2 は存在しない（表中斜線の部分）．また，厳密には予測バグ数を求める事ができないため, バグが含まれる確率を予測バグ数の代わりに用いる事とする. 以下では，バグの有無を目的変数とした場合では，テスト戦略 T0,T1,T3,T4 での総バグ発見率の比較を行う事とする.

目的変数ごとの総バグ発見率はテスト戦略 T1（予測値に比例させたテスト工数割り当て戦略）の基で比較すると，バグ密度を目的変数としてモデルを構築し

た場合が最も総バグ発見率が高くなった（77%）．同様に，テスト戦略 T3,T4 の基で比較すると T3 の基ではバグの有無を目的変数としたときであり, T4 ではバグ密度を目的変数とした時が最も総バグ発見率が高くなった. 最も総バグ発見率が高いのはバグ密度を目的変数としてモデルを構築し，テスト戦略 T1，T4 でテスト工数割り当てを行った時であった（総バグ発見率：77%）．

表 5 各目的変数での総バグ発見率

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

4.2.3 RQ2 に対する実験結果

RQ2 に対する実験結果を表 6 に示す．表中の値は各バージョン間予測の組み合わせでの最も総バグ発見率の高いテスト戦略を表している．バグの有無を目的変数としてモデルを構築した場合の結果については，テスト戦略 T2 は評価対象外としている．表中の斜線部分は表 4 において AUC もしくは P_{opt} の値が 0.7 未満のため評価対象外のモデルである事を示す．表 6 より，8 つの予測結果に基づくテスト工数割り当てで，T4：予測バグ数×対数規模に比例させたテスト工数割り当て戦略が最も総バグ発見率の高いテスト戦略になっている．また，表 5 より各テスト戦略での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（総バグ発見率：76% ）．よって，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．

表 6 総バグ発見率の高いテスト戦略

		目的変数					
		バグの有無		バグ数		バグ密度	
バージョン		LR	RF	LM	RF	LM	RF
v3.0	v3.1	T3	T4	T0	T1	T4	
v3.0	v3.2	T4	T3	T4	T4	T4	T4
v3.1	v3.2	T3	T3		T4	T1	

（LR:ロジスティック回帰 LM:重回帰 RF:ランダムフォレスト）

5 おわりに

本論文では，バグモジュール予測の予測結果に基づいたモジュールへのテスト工数割り当ての効果と有用性を示す事を目的として，Eclipse プロジェクトから収集したデータセットを用いてテスト工数割り当てのシミュレーション実験を行った.

本論文で得られた知見は下記の 2 つである．

- 予測バグ数と対数規模を掛け合わせた値に比例させてテスト工数をモジュールに割り当てる事で，多くのバグを発見できる．
- 予測バグ数を求めるにあたっては，バグ数を直接予測するのではなく，バグ密度を予測してから規模を乗じてバグ数を算出することが望ましい．

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できると考えられる．
今後の課題として，バグの重要度を考慮したシミュレーションを行うことが考えられる．例えば，バグの重要度ごとにバグ数（またはバグ密度）を予測し，目標とするバグ発見率をそれぞれ定めることで，必要となるテスト工数をシミュレーションにより見積もることが考えられる．また，本論文では実験対象として OSS の 1 プロジェクトのみに対して実験を行い，バグモジュール予測結果を用いたテスト工数割り当ての効果について明らかにした．より妥当性を高めるために他のプロジェクトデータに対しても分析を進める必要がある．

謝辞

研究の一部は，文部科学省科学研究費補助金（基盤研究(C)：課題番号 22500028）に基づいて行われた．

6 参考文献

[1] M. D'Ambros, M. Lanza and R. Robbes: An extensive comparison of bug prediction approaches, In Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR2010), pp. 31-41, 2010.

[2] P. Knab, M. Pinzger and A. Bernstein: Predicitng defect densities in source code files with decision tree learners, In Proceedings of the 3rd Working Conference on Mining Software Repositories (MSR2006), pp. 119-125, 2006.

[3] S. Lessmann, B. Baesens, C. Mues and S. Pietsch: Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Transactions on Software Engineering, Vol.34, No.4, pp.485-496, 2008.

[4] P. Li, J. Herbsleb, M. Shaw and B. Robinson: Experiences and Results from Initiating Field Defect Prediction and Product Test Prioritization Efforts at ABB Inc, In Proceedings of the 28th International Conference on Software Engineering(ICSE2006), pp.413-422, 2006.

[5] T. McCabe: A complexity measure, IEEE Transactions on Software Engineering, Vol. 2, pp.308-320, 1976.

[6] T. Mende and R. Koschke: Revisiting the evaluation of defect prediction models, In Proceedings of the International Conference on Predictor Models in Software Engineering (PROMISE2009), pp.1-10, 2009.

[7] N. Nagappan, T. Ball and A. Zeller: Mining Metrics to Predict Component Failures, In Proceedings of the 28th International Conference on Software Engineering(ICSE2006), pp.452-461, 2006.

[8] N. Ohlsson, H. Alberg: Predicting fault-prone software modules in telephone switches,

IEEE Transactions on Software Engineering, Vol.22, No.12, pp.886-894, 1996.

[9] B. Turhan, T. Menzies, A. Bener and J. Distefano: On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering, Vol. 14, No. 5, pp. 540-578, 2009.

[10] 柿元 健, 門田 暁人, 亀井 靖高, まつ本 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り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, pp.1716-1724, 2009.

[11] 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* 2011, 日経 BP , 2011.

[12] 山田茂: *ソフトウェア信頼性モデル−基礎と応用*, 日科技連出版社, 1994.