学習データ計測時点による欠陥モジュール予測精度の比較

Comparison of Faulty Module Prediction Accuracies with Different Fit Data Measurement Points

内垣　聖史* 伊原　彰紀† 門田　暁人‡ 松本　健一§

あらまし 近年，ソフトウェア開発におけるテスト工程の効率化を目的として，欠陥が含まれている可能性の高いモジュールを予測するための研究 (欠陥モジュール予測) が行われてきた. 従来の欠陥モジュール予測では，モデル構築に用いる学習データとして過去のバージョン開発終了時のデータが使用されていた．昨今，反復型開発モデルを導入するケースが増加しているため，開発途中であっても学習データの計測が可能となった．本論文では，学習データ計測時点による予測精度の違いを分析する．分析の結果，学習データとして過去バージョン開発終了時のデータを用いるより，予測時点に近い開発途中のデータを用いた方が精度が高いことが分かった．

1　はじめに

ソフトウェア開発プロセスのテスト工程において，欠陥を含んでいる可能性の高いモジュール（本論文ではソースファイルを対象とする）を特定することは，テストの効率化のために重要である [1]．ソフトウェアの大規模化に伴い全モジュールに対してテストを行うことはコスト面及び時間的に困難であるため，重点的にテスト工数を割り当てるべき欠陥モジュールを特定する (欠陥モジュール予測) 技術が提案されている [2] [3]．欠陥モジュール予測手法により，同じ工数でもより多くの欠陥を検出できテスト工数の効率化が可能になる [4].

従来，ウォータフォール型の開発プロジェクトを対象とした欠陥モジュール予測では，予測対象プロジェクトにおける過去バージョンの開発終了時のスナップショット[1]を用いて予測モデルが構築されることが多い．その理由は，ウォーターフォール型開発では開発後期になるまでコーディングが始まらないため，予測モデルの構築に必要となるモジュールのメトリクス値を開発途中で計測できないからである．たとえモジュール以外の情報 (クラス間の参照関係，クラスの属性) を使用して開発途中に予測モデルを構築したとしても，予測精度は低くなる [5].

一方で，反復型開発モデルのように短いサイクルで機能を拡張していく開発手法では，開発途中であっても欠陥モジュール予測に必要なメトリクス (コード行数やサイクロマティック数，分岐数) を取得することができる．従って，必ずしも開発終了時のスナップショットを用いて予測モデルを構築する必要はない.

本論文では，開発途中のスナップショットを用いて欠陥モジュール予測モデルを構築し，学習データの取得時点による予測精度の違いを分析する．実験では実際のテスト工程での利用を想定し，従来から多く研究されてきた欠陥の有無ではなく，欠陥密度を予測するモデルで性能を評価する．欠陥密度を予測対象とする場合，欠陥有無の予測とは異なりモジュール中の欠陥数やモジュール規模が考慮可能となる．そのため，同様に欠陥が含まれているモジュールであってもより欠陥が多いモジュー

*Satoshi Uchigaki, 奈良先端科学技術大学院大学

†Akinori Ihara, 奈良先端科学技術大学院大学

‡Akito Monden, 奈良先端科学技術大学院大学

§ken-ichi Matsumoto, 奈良先端科学技術大学院大学

[1]スナップショット:ある時点のソフトウェアを構成するモジュールの集合

ルに対してより多くの工数を割り当てることが可能になる等，テスト工程をより効率化することができる．本論文で実施した実験によって，欠陥モジュール予測の実施に当たり学習データとして利用すべき (最も予測精度が高い) スナップショットの取得時点を明らかにすることができる．

以降，2 章で従来の欠陥モジュール予測手法を説明する．3 章では，欠陥モジュール予測実験について述べ，4 章で実験結果について議論する．最後に 5 章で本論文のまとめと今後の課題を述べる．

2　欠陥モジュール予測

欠陥モジュールを特定するために，現在までに多数の欠陥モジュール予測モデルが用いられている [6] [7] [8]．欠陥モジュール予測モデルは，過去に開発されたプロジェクトのモジュールに関するメトリクスを説明変数とし，モジュールに含まれる欠陥情報 (欠陥の有無や欠陥密度 (単位行数あたりの欠陥数)) を目的変数として構築される [2] [3] [9]．構築した予測モデルに対して新規に開発したプロジェクトのモジュールから計測したメトリクスの値を入力することで，そのモジュールに欠陥が含まれる確率や欠陥密度を予測することができる．特に近年，欠陥モジュール予測モデルには汎化性や予測精度の高さから Breiman [10] によって提唱されたランダムフォレストがよく利用されている [11].

2.1　開発終了時のデータを用いた欠陥モジュール予測

反復型開発プロジェクトを対象とした欠陥モジュール予測について述べられている論文は少なくない．亀井ら [12] は粗粒度モジュールに対する欠陥密度予測を目的として Eclipse プロジェクトを用いた評価実験を行っている．水野ら [13] は Eclipse TPTP plugin 及び Eclipse BIRT plugin の各プロジェクトに対して，モジュールのスパムフィルタリングによる欠陥の判別を実施している．しかしながら両研究とも反復型開発プロジェクトを対象としているが，いずれも開発終了時点のスナップショットから予測モデルを構築しており開発途中のスナップショットを学習データとして使用していない．

2.2　開発途中のデータを用いた欠陥モジュール予測

村尾ら [14] は開発の進行に伴うメトリクス値の変遷に基づき，メトリクス値の変動がどの程度安定しているかを表す指標である恒常性を提案し，恒常性の低いモジュールは今後欠陥の修正対象になりやすいことを実験により示した．時期によって開発されるモジュールが異なる反復型開発プロジェクトに，村尾らが提案する予測手法を用いる場合，恒常性を計測する期間に変更されたモジュールのみが対象となる．多くのモジュールを対象とする場合，恒常性を計測する期間を長期間にする必要があるため，バージョン管理システムから多くのスナップショットの取得に時間を要する．

2.3　本論文の目的

本論文の目的は，開発途中のスナップショットを学習データとして欠陥モジュール予測モデルを構築し，学習データの取得時点による欠陥モジュールの予測精度の違いを分析することである．本論文により，実際に欠陥モジュール予測を実施する上で，どの段階のスナップショットを学習データとして利用し予測モデルを構築すればより高い精度で欠陥モジュールを予測することができるのかを明らかにできると考えられる．

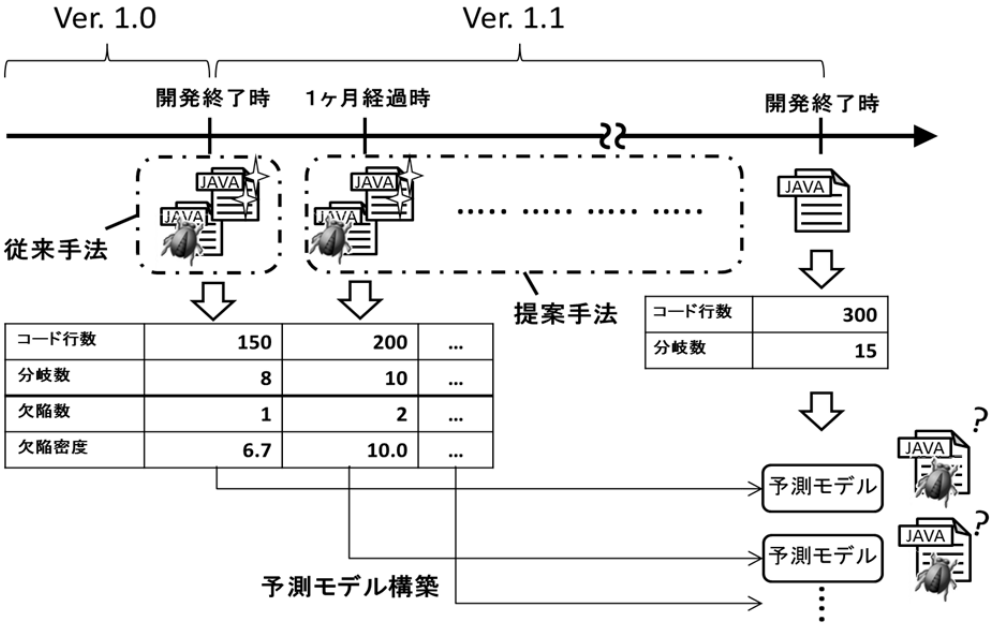

図 1　実験の概略図

3　実験

3.1　概要

本論文の実験では，予測モデルの構築に利用する学習データの取得時点が欠陥モジュール予測モデルの予測精度に影響を与えるか否かを明らかにする．具体的には，まず開発途中の各時点のスナップショットから計測されたメトリクス，及び，本論文の比較対象として過去バージョンの開発終了時のスナップショットから計測されたメトリクスを学習データとしてそれぞれ予測モデルを構築する．各学習データには各月 1 日のスナップショットを利用し，学習データの取得時点による予測精度の違いを分析する．本実験では実際のテスト工程での利用を想定し，従来から多く研究されてきた欠陥の有無ではなく欠陥密度を予測する．欠陥密度を予測対象とする場合，欠陥有無の予測とは異なりモジュール中の欠陥数やモジュール規模が考慮可能となる．そのため，同様に欠陥が含まれているモジュールであってもより欠陥が多いモジュールに対してより多くの工数を割り当てることが可能になる等，テスト工程をより効率化することができる．実験の概略図を図 1 に示す．本論文ではモデル構築アルゴリズムとして，近年，精度や汎化性の高さから注目されているランダムフォレスト [10] を採用する．また本論文では 1 つのソースファイルを 1 モジュールとして扱う．

3.2　データセット

本研究では，Eclipse プロジェクトの 2005 年 7 月 (Eclipse ver.3.1 リリース時相当) から 2006 年 7 月 (Eclipse ver.3.2 リリース時相当) までの 13 か月分 (各月 1 日) のスナップショットを用いて欠陥モジュール予測モデルを構築し，実験を行った．各スナップショットは Eclipse プロジェクトが管理するバージョン管理システム CVS[2] に

[2]Eclipse CVS: http://home.segal.uvic.ca/~schadr/msrc2011/eclipse-cvs.tgz (2013 年 7 月 4 日現在，サイトからデータを取得できない．)

保存された対象プロジェクトの変更履歴情報を利用して各月ごとに取得した．実験に利用したEclipseプロジェクトの概要を表1に示す．また各スナップショットの概要を表2に示す．表2の欠陥数と欠陥モジュール数は，各スナップショットの取得対象時点から3か月間に報告された欠陥及び欠陥が含まれていたモジュール数である．

本論文ではモジュール単位でのプロダクトメトリクス及びプロセスメトリクスを計測した．各メトリクスの概要を表3に示す．プロダクトメトリクスはソースコード構造解析ツール「Understand」[3]を用いてモジュールからコード行数，クラス数など10種類のメトリクスを収集した．プロセスメトリクスはバージョン管理システムにおけるコミットログから，モジュールの過去3か月分の変更回数，追加行数，削除行数の3種類のメトリクスを収集した．開発途中でモジュールが作成された場合，作成後3か月間は3か月分の変更量を計測することができないため，全スナップショットに共通するモジュールのみを使用する．本論文では前述した計13種類のメトリクスを説明変数として予測モデルを構築する．

本論文ではSZZアルゴリズム [15] を用いて欠陥を含むモジュールを特定した．SZZアルゴリズムはバグ管理システムに記録された欠陥情報 (欠陥の報告日時，コメント，バグID) とバージョン管理システムに記録されたモジュールの編集履歴 (コミット日時，コミット者，コメント) を対応付けることで欠陥が混入したモジュールや混入日時を特定する手法である．

表 1　Eclipse プロジェクトの概要

開発言語	Java
総スナップショット数	13
スナップショットの取得対象期間	2005/07/01 〜2006/07/01
各スナップショットの取得対象時点	各月の始め (e.g. 2005/12/01 00:00:00)
スナップショットのモジュール数 (共通)	3415

表 2　スナップショットの概要

取得時点	欠陥モジュール数	欠陥モジュール率 [%]	欠陥数	総行数 [KLOC]
2005/07/01	342	10.0	396	538
2005/08/01	538	15.8	845	541
2005/09/01	416	12.2	665	543
2005/10/01	436	12.8	698	546
2005/11/01	467	13.7	797	550
2005/12/01	940	27.5	1,423	553
2006/01/01	1,037	30.4	1,532	554
2006/02/01	1,211	35.6	1,741	557
2006/03/01	650	19.0	963	566
2006/04/01	535	15.7	758	569
2006/05/01	223	6.5	321	572
2006/06/01	310	9.0	481	572
2006/07/01	370	11.0	610	574

[3] http://www.techmatrix.co.jp/quality/understand/

表 3　メトリクスの概要

	メトリクス名	概要
プロダクト	AvgCyclomatic	メソッドのサイクロマティック複雑度の平均
メトリクス	CountDeclClass	クラス数
	CountDeclFunction	関数の数
	CountLineBlank	空白行数
	CountLineCode	コード行数
	CountLineCodeDecl	宣言コード行数
	CountLineCodeExe	実行可能コード行数
	CountLineComment	コメント行数
	CountStmtDecl	宣言ステートメント行数
	CountStmtExe	実行可能ステートメント行数
プロセス	Number of revisions	変更回数
メトリクス	Added lines	追加行数
	Removed lines	削除行数

3.3　評価指標

本論文では予測モデルの評価指標として Mende [16] らの提案した P_{opt} を用いる．P_{opt} は，欠陥密度の予測値が大きい順にモジュールを抽出した際に実際に抽出できた欠陥の積み上げグラフ (図 2) の曲線下面積から計算される．具体的には，欠陥密度の実測値が大きい順に並べた際の積み上げグラフ (図 2 の上のグラフ) の曲線下面積とモデルの予測値による積み上げグラフ (図 2 の下のグラフ) の曲線下面積の差分から求められ，式 (1) で定義される．また P_{opt} の値域はデータの欠陥数によって変動するため，本論文では式 (2) により正規化を行う．ここで $MAX(P_{opt})$，$MIN(P_{opt})$ はそれぞれ P_{opt} の取り得る最大値，最小値である．正規化後の P_{opt} の値域は [0,1] であり，値が 1 に近づくほど予測精度が高いことを示す．以降，本論文で表記する P_{opt} は正規化されたものを用いる．P_{opt} は，従来欠陥モジュール予測の評価指標として広く採用されていた適合率や再現率，F1 値とは異なり，精度が判別結果を 2 値に区切る際の閾値に影響されない．似た評価指標として Alberg Diagram [17] の AUC があるが P_{opt} では AUC と異なり，モジュールの規模を考慮した評価が可能である．

$$P_{opt} = 1 - \Delta_{opt} \tag{1}$$

$$\text{正規化 } P_{opt} = \frac{P_{opt} - MIN(P_{opt})}{MAX(P_{opt}) - MIN(P_{opt})} \tag{2}$$

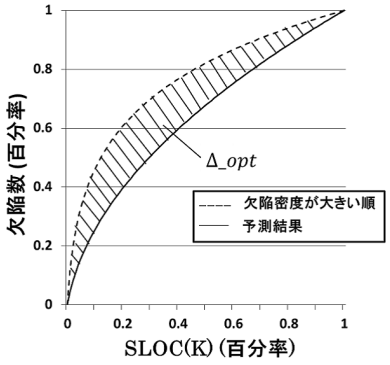

図 2　積み上げグラフ

3.4 実験手順

実験は以下に示す 3 つの手順で構成される．

手順 1: 予測モデルの構築

スナップショットの内，取得時点が最も遅いスナップショット (2006 年 7 月時) を評価データとし，その他のスナップショットを学習データとしてそれぞれランダムフォレスト法を用い，13 種のメトリクスを説明変数，欠陥密度を目的変数とする予測モデルを構築する．

手順 2: 欠陥密度の予測

構築した 12 個の予測モデルにそれぞれ評価データのメトリクス値を適用し，各モジュールの欠陥密度を予測する．

手順 3: P_{opt} の算出

各モデルから得られた欠陥密度の予測値と評価データから計測した欠陥密度の実測値から P_{opt} をそれぞれ算出する．

3.5 実験結果

各学習データごとの実験結果を図 3 に示す[4]．横軸は予測モデルを構築したスナップショットの取得時点，縦軸は P_{opt} の値を示している．図 3 より，取得時点が予測時点 (2006 年 7 月時) から最も遠いスナップショット (2005 年 7 月時) から構築されたモデルを利用した場合に最も予測精度が低くなり，一方で最も近いスナップショット (2006 年 6 月) の場合に最も予測精度が高いことが分かった．なお，取得時点が予測時点から最も遠いスナップショットから構築したモデルでは，予測値 (欠陥密度) の高い上位 50 %のモジュールから発見された欠陥の 67 %を検出した．一方で，最も予測時点に近いスナップショットでは，83 %の欠陥を検出できたため，開発終了時に近い時点のモジュールから学習データを作成した方が，高い予測精度を得るこ

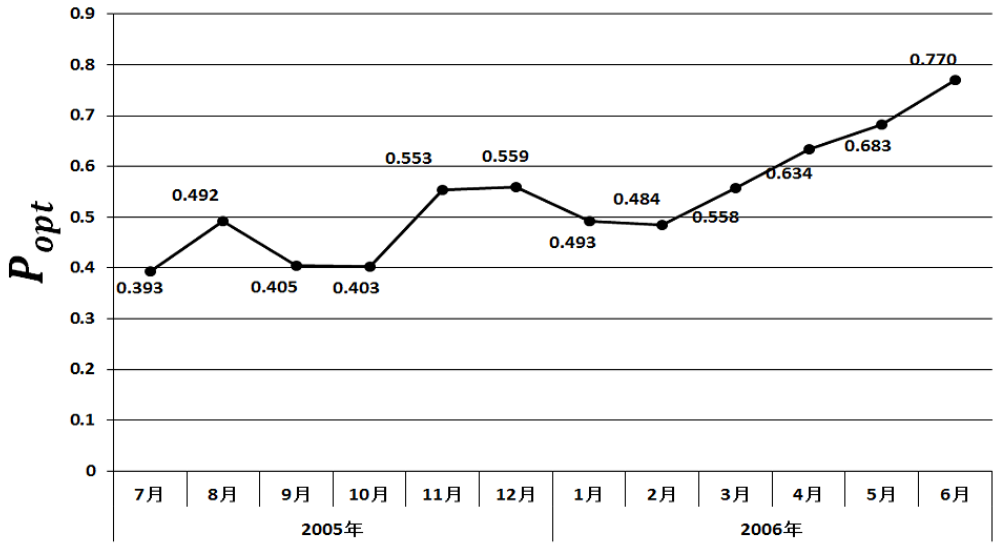

図 3 各スナップショットを学習データとした場合の P_{opt} の推移 (評価データ：2006 年 7 月)

[4]本論文が対象とする欠陥は，各スナップショットの取得対象時点から 3 か月以内にバグ管理システムに記録されたものであるため，評価データから直近 2 か月分のスナップショット (2006 年 5 月及び 2006 年 6 月時点) は本来学習データとしては利用できない．本論文では学習データの取得時点と欠陥の予測精度の間の関係性の分析を目的とするため，例外的に評価データ以後に記録された欠陥を含めた 3 か月以内の欠陥を利用している．

とができ，多くの欠陥を検出することができるといえる．また，予測精度の時系列推移を最小二乗法を用いて傾向推定した結果，スナップショットの取得時点が遅くなるほど予測精度が上昇する傾向にあった (有意水準 p<0.01)．

本論文では，この結果を基に「学習データ及び評価データとして利用するスナップショットの取得時点が近いほど予測精度が高くなる」という仮説を立案し，仮説の検証のために追加実験を実施する．

3.6 追加実験

前述の実験では，評価データをリリース時で固定し (2006 年 7 月)，学習データの取得時点を変えた結果，学習データ，及び評価データの取得時点が近いほど予測精度が高い傾向が見られた．本節では学習データと評価データの取得時点が近ければ予測精度が向上することを確認するために追加実験を実施する．追加実験では学習データとなるスナップショットの取得時点を固定し (本追加実験では 2005 年 7 月時点)，評価データは，学習データ以後 (2005 年 8 月から 2006 年 7 月まで) のスナップショットとし，評価データ取得時点による予測精度の違いを分析する[5]．追加実験の概略図を図 4 に示す．予測モデル構築には，表 3 に示すメトリクスを用いる．追加実験の結果を図 5 に示す．追加実験の結果，前述の実験と同様，評価データと学習データの取得時点が近いほど予測精度が高くなる傾向が見られた (有意水準 p<0.01)．よってこの結果は前節で立案した仮説を支持するものである．

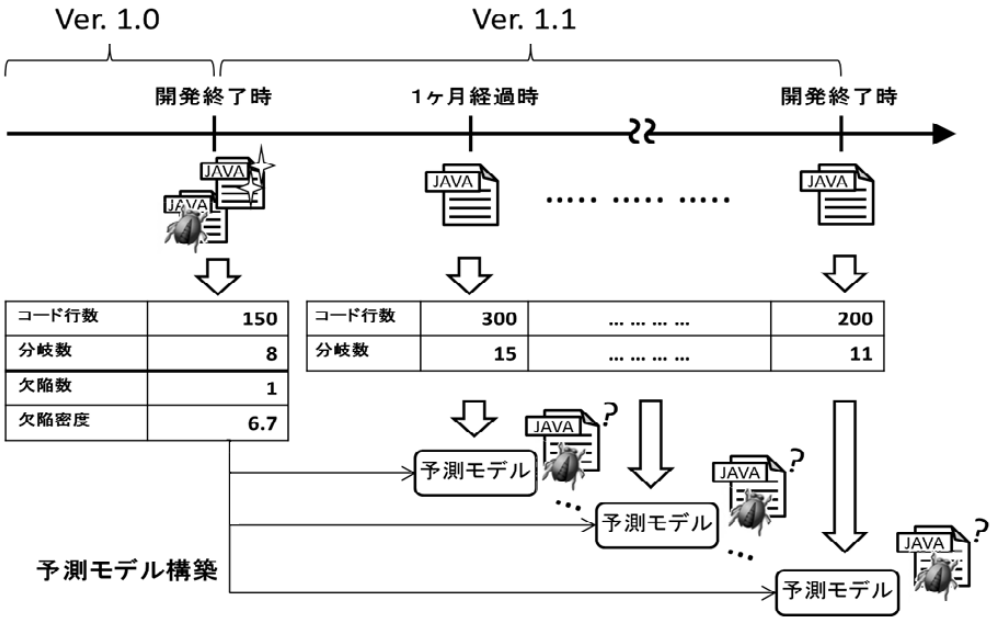

図 4　追加実験の概略図

[5]実際の欠陥モジュール予測では基本的に開発終了時のモジュールの欠陥予測を目的とするため開発途中のスナップショットデータを評価データとすることはない．本論文では，仮説の検証のためにあえて開発中のデータを評価データとして扱っている．

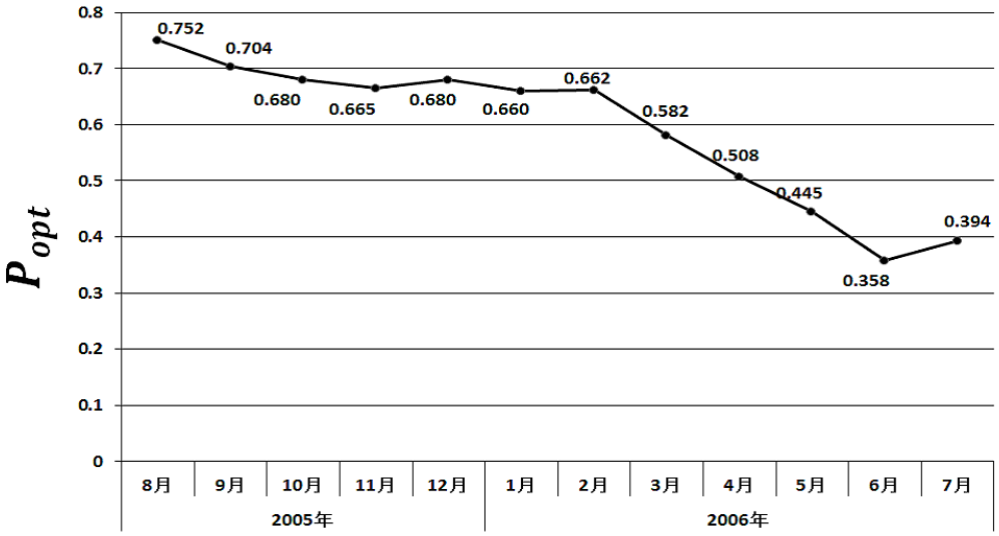

図 5　各スナップショットを評価データとした場合の P_{opt} の推移 (学習データ：2005 年 7 月)

表 4　スナップショット別のメトリクスの値域 (太字は値域の変動が大きいメトリクス)

	2005 年		2006 年		
メトリクス	7 月	10 月	1 月	4 月	7 月
AvgCyclomatic	[0 , 19]	[0 , 19]	[0 , 19]	[0 , 19]	[0 , 19]
CountDeclClass	[1 , 65]	[1 , 65]	[1 , 65]	[1 , 65]	[1 , 65]
CountDeclFunction	[0 , 1,849]	[0 , 1,881]	[0 , 1,929]	[0 , 2,051]	[0 , 2,243]
CountLineBlank	[0 , 660]	[0 , 660]	[0 , 660]	[0 , 661]	[0 , 665]
CountLineCode	[3 , 8,256]	[3 , 8,421]	[3 , 8,628]	[3 , 9,151]	[3 , 10,032]
CountLineCodeDecl	[2 , 2,332]	[2 , 2,354]	[2 , 2,381]	[2 , 2,584]	[2 , 2,713]
CountLineCodeExe	[0 , 4,282]	[0 , 4,357]	[0 , 4,467]	[0 , 4,747]	[0 , 5,227]
CountLineComment	[10 , 3,372]	[10 , 3,365]	[10 , 3,419]	[10 , 3,410]	[10 , 3,342]
CountStmtDecl	[2 , 2,331]	[2 , 2,352]	[2 , 2,380]	[2 , 2,583]	[2 , 2,712]
CountStmtExe	[0 , 3,436]	[0 , 3,496]	[0 , 3,584]	[0 , 3,808]	[0 , 4,192]
Number of revisions	[0 , 63]	[0 , 36]	[0 , 35]	[0 , 69]	[0 , 26]
Added lines	**[0 , 1,329]**	[0 , 1,579]	[0 , 1,366]	[0 , 3,096]	**[0 , 3,223]**
Removed lines	**[-816 , 0]**	[-1,465 , 0]	[-1,181 , 0]	[-2,919 , 0]	**[-3,237 , 0]**

4　考察

4.1　学習データ取得時点による予測精度の違いに関する妥当性の考察

実験において，学習データ及び評価データとして利用するスナップショットの取得間隔が短ければ，欠陥密度の予測精度が高く，間隔が長ければ精度が低い傾向にあるという結果が得られた．その理由として，通常，ソフトウェア開発においては開発が進むにつれてコード行数は増加するため，学習データと評価データの取得時点が離れるほどメトリクスの分布が異なってくるからと考えられる．一般的に，機械学習において学習データと評価データが類似している場合，予測精度は高くなり，各データが異なるほど予測精度が低下する [18]．実験において各スナップショットの取得間隔が長いほど精度が低くなった理由として，時間の経過とともに徐々にモジュールが変更され，評価データが予測モデルに適合できなくなったことが考えら

れる．本節ではスナップショット取得時点によるメトリクスの違いを分析する．スナップショットには複数のモジュールが含まれているため，スナップショットに含まれる全モジュールから取得したメトリクスの値域を比較する．

本実験で取得したスナップショットの各メトリクスの値域を表 4 に示す．本論文では紙面の都合上，3 か月おきのスナップショットのメトリクスの値域を記載する．表 4 より，Number of revisions(変更回数) を除く 12 個のメトリクスに関して，スナップショット間でほとんど変化が見られないか，あるいは古いスナップショットに比べ新しいスナップショットの方が値域が拡大していることが分かる．特に 2006 年 7 月時における Added lines(追加行数) の値域は 2005 年 7 月と比べ約 2.4 倍，Removed lines(削除行数) の値域は約 4 倍と非常に差があることが分かる．この結果は，時間の経過と共にモジュールが徐々に変更され，予測モデルに評価データが適合しにくくなるという筆者らの考えを裏付けるものである．よって実験での予測精度には学習データと評価データのメトリクスの値域差が関係していることが考えられる．また表 4 で，追加行数や削除行数が大きいのにも関わらずプロダクトメトリクスは大きく変化していない．この理由は，新規機能の追加や削除に比べ，既存機能の変更（変数名の変更，コメントの追加/削除など）が多かったと考えられる．

4.2 予測精度に関する妥当性の考察

Khoshgoftaar ら [19] によると学習データに含まれる欠陥モジュール数の割合が欠陥のないモジュール数と比べて著しく少ない場合は性能の良いモデルを構築することが困難な場合がある．本論文で使用したスナップショットの総モジュール数に対する欠陥モジュール数の割合は，6.5 %〜35.6 % (中央値 13.7 %) であり，欠陥モジュール数の割合が欠陥のないモジュール数に比べ少ないといえる．また実際の開発環境では開発が進むにつれて，全モジュール中の欠陥モジュールの割合は小さくなることが考えられる．このためサンプリング法 [20] 等を適用してスナップショット中の欠陥モジュール数と欠陥のないモジュール数との割合の極端な偏りを改善することにより，更なる予測精度の向上が期待できると考えられる．

5 おわりに

本論文では，Eclipse プロジェクトから収集した 13 か月分のスナップショットデータを対象に，開発途中のスナップショットを用いて欠陥モジュール予測モデルを構築し，学習データの取得時点による予測精度の違いを分析した．また実験で得られた欠陥密度の予測精度と各スナップショットのメトリクスの値域の関係性について考察を行った．実験及び考察を通して，以下の知見が得られた．

- 学習データ及び評価データとして利用する各スナップショットの取得間隔が短いほど高い予測精度が得られる．
- 予測精度の高低は，各スナップショット間のメトリクスの値域の違いに大きく左右される．

本論文で得られた知見を用いて，開発後期のスナップショットデータを学習データとして予測モデルを構築することにより，管理者はテスト工程で重点的に工数を割り当てるべきモジュールをより精度よく予測することができると考えられる．今後はメトリクス値の変動が大きいモジュールのみを対象とした予測実験を行い，更に本実験で得られた知見の妥当性を向上させるために他のプロジェクトデータを用いた実験を行う．

謝辞

本研究の一部は，文部科学省科学研究補助費（若手 (B): 25730045）および（基盤 (B): 23300009）による助成を受けた．また，本研究の一部は，公益財団法人中島記念国際交流財団による助成を受けた．

参考文献

[1] 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. International Conference on Software Engineering (ICSE’06), pp.413-422, 2006.

[2] N. Nagappan, T. Ball, and A. Zeller, “ Mining metrics to predict component failures, ” Proc. International Conference on Software Engineering (ICSE’06), pp.452-461, 2006.

[3] 纐纈 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, Vol.2009-SE-168, No.6, pp.1-8, 2009.

[4] 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.

[5] T. Kamiya, S. Kusmoto, and K. Inoue, “ Prediction of fault proneness at early phase in object-oriented development, ” Proc. International Symposium on Object-Oriented Real Time Distributed Computing (ISORC’99), pp.253-258, 1999.

[6] J.C. Munson, and T.M. Khoshgoftaar, “ The detection of fault-prone programs, ” IEEE Trans. Software Engineering, Vol. 18, No. 5, pp. 423-433, 1992.

[7] M. Pighin, and R. Zamolo, “ A predictive metric based on statistical analysis, ” Proc. International Conference on Software Engineering (ICSE’97), pp.262-270,1997.

[8] 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, pp.297-316, 1999.

[9] まつ本 真佑, 亀井 靖高, 門田 暁人, 松本 健一, “ Fault-Prone モジュール判別モデルに対する外れ値除去法の適用効果, ” 情報処理学会研究報告, ソフトウェア工学, Vol.155, pp.49-56, 2007.

[10] L. Breiman, “ Random forests, ” Machine Learning, Vol.45, No.1, pp. 5-32, 2001.

[11] K. Kawamoto, and O. Mizuno, “ Predicting fault-prone modules using the length of identifiers, ” Proc. International Workshop on Empirical Software Engineering in Practice (IWESEP’12), pp.30-34, 2012

[12] 亀井 靖高, まつ本 真佑, 門田 暁人, 松本 健一, “ 粗粒度モジュールに対するバグ密度予測の精度評価, ” 電子情報通信学会技術研究報告, Vol.109, No.456, pp.145-150, 2010.

[13] O. Mizuno, and H. Hata, “ Prediction of fault-prone modules using a text filtering based metric, ” International Journal of Software Engineering and Its Application, Vol.4, No.1, pp.43-52, 2010.

[14] 村尾 憲治, 肥後 芳樹, 井上 克郎, “ ソフトウェアメトリックス値の変遷に基づいた注力すべきモジュールを特定する手法の提案, ” 電子情報通信学会論文誌, Vol.J91-D, No.12, pp.2915-2925, 2008.

[15] J. Sliwerski, T. Zimmermann, and A. Zeller, “ When do changes induce fixes?, ” Proc. International Workshop on Mining Software Repositories (MSR’05), pp.24-28, 2005.

[16] T. Mende, and R. Koschke, “ Revisiting the evaluation of defect prediction models, ” Proc. International Conference on Predictor Models in Software Engineering (PROMISE’09), pp.1-10, 2009.

[17] N. Ohlsson, and H. Alberg, “ Predicting fault-prone software modules in telephone switches, ” IEEE Trans. Software Engineering, Vol.22, No.12, pp.886-894, 1996.

[18] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, “ Cross-project defect prediction, ” The 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’09), pp.91-100, 2009.

[19] T.M Khoshgoftaar, K. Gao, and R.M. Szabo, “ An application of zero-inflated poisson regression for software fault prediction, ” Proc. International Symposium on Software Reliability Engineering (ISSRE’01), pp.66-73, 2001.

[20] 亀井 靖高, まつ本 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.