社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

プロセスメトリクスを用いたメソッド抽出事例の調査と予測モデルの構築

田中　大樹† 　崔　　恩瀞† 　吉田　則裕†† 　藤原　賢二††† 　飯田　　元†

† 奈良先端技術大学院大学情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
†† 名古屋大学大学院情報科学研究科　〒 464–8601 名古屋市千種区不老町
††† 豊田工業高等専門学校情報工学科　〒 471–8525 愛知県豊田市栄生町 2–1
E-mail: †{tanaka.daiki.sx4,choi}@is.naist.jp, ††yoshida@ertl.jp, †††fujiwara@toyota-ct.ac.jp, ††††iida@itc.naist.jp

あらまし　メソッド抽出は，既存のメソッドの一部を新規メソッドとして抽出するリファクタリング手法の一つであり，実施される回数が多いことも知られている．メソッド抽出を支援するためには，開発者がどのようなメソッドを抽出の対象としているかを調査する必要がある．本研究では，未だ定量的な調査が実施されていない，ソフトウェアの開発履歴から計測されるプロセスメトリクスとメソッド抽出の関係を調査した．調査の結果，いくつかのプロセスメトリクスとメソッド抽出の実施に相関がみられた．また，調査によって有効性が確認されたプロセスメトリクスを利用して，メソッド抽出の候補となるメソッドを推薦する機械学習モデルを構築し，その精度を評価した．実験の結果，プロダクトメトリクスとプロセスメトリクスの両方を利用した推薦モデルが，F 値で 0.827 と最も高い精度を示した．
キーワード　メソッド抽出リファクタリング，プロセスメトリクス，リファクタリング推薦，ソフトウェア保守

An Investigation of Extract Method to Construct Prediction Model Based on Process Metrics

Daiki TANAKA†, Eunjong CHOI†, Norihiro YOSHIDA††, Kenji FUJIWARA†††, and Hajimu IIDA†

† Graduate School of Information Science, Nara Institute of Science and Technology　8916–5, Takayama, Ikoma, Nara, 630–0192,Japan
†† Graduate School of Information Science, Nagoya University　Furo-cho, Chikusa-ku, Nagoya, 464–8601, Japan
††† Information and Computer Engineering, National Institute of Technology, Toyota College　2–1 Eiseicho, Toyota, Aichi, 471–8525, Japan
E-mail: †{tanaka.daiki.sx4,choi}@is.naist.jp, ††yoshida@ertl.jp, †††fujiwara@toyota-ct.ac.jp, ††††iida@itc.naist.jp

Abstract　Extract method is one of the most widely used refactoring patterns. For efficient refactoring support, we need to investigate methods that developers perform extract method. In this study, we investigated a relationship between process metrics and extract method. As the result, we confirmed a correlation on process metrics between extract method activities. Furthermore, we constructed and evaluated six machine learning models using process metrics to recommend extract method refactoring. The result shows that the highest F-score is 0.827 in the case of the model that combines both process and product metrics.
Key words　Extract Method Refactoring，Process Metrics，Refactoring Recommendation，Software Maintenance

1. はじめに

リファクタリングとは，外部から見たときの振る舞いを保ちつつ，保守性や可読性を向上させるために，ソフトウェアの内部構造を整理することである [1]．数多くのリファクタリングパターンが定義されており，どのようなコード片に対してどのリファクタリングパターンを実施すべきかが示されている．

開発者はソフトウェア開発や保守において，ソースコードの保守性や可読性を向上させるために，リファクタリングを実施することが求められるが，開発者が大規模なソースコードからリファクタリング候補を見つけることは困難である．そのため，開発者のリファクタリング作業を支援するための技術やツールが多く提案されている [2] [3]．

しかし，これらの技術やツールを，開発者はあまり利用していないという報告もあるため [4]，研究者は開発者のリファクタリング作業の実態をより正確に把握したうえで，支援手法を検討していく必要がある．後藤らは，リファクタリングパターンの 1 つである，メソッド抽出リファクタリングの対象となったメソッドの特徴を調査した [5]．具体的には，抽出対象となったメソッドのサイズや凝集度などのプロダクトメトリクスを計測し，抽出が行われなかったメソッドとの比較調査をしている．

ソースコードの特徴を定量的に表現する方法として，プロダクトメトリクス以外に，開発の履歴情報を用いて計測するプロセスメトリクスがある [6]．開発プロセスはソースコードの品質に影響を与える．リファクタリングは一般的に，保守性や可読性が低い品質の悪いコード片を対象に実施されるため，ソースコードの品質に影響を与える開発プロセスから計測されるプロセスメトリクスと，リファクタリングの実施には相関があると考えられる．しかし，プロセスメトリクスとリファクタリングの関係について，定量的な調査は実施されていない．

そこで，本研究ではプロセスメトリクスを用いて，メソッド抽出の対象となったメソッドの特徴を調査した．ソースコードの品質に影響を与えていると考えられる，コード片の変更回数や，その変更に関わった開発者数等のメトリクスを用いて抽出の対象となったメソッドと，対象とならなかったメソッドを比較した．その結果，コード片の変更回数とオーサー数，コード片の変更量である追加・削除・変更行数に，メソッド抽出の対象となったメソッドと，対象とならなかったメソッド間で有意差が確認できた．

また，プロセスメトリクスの調査結果を踏まえ，メソッド抽出の対象となるメソッドを推薦する機械学習モデルを構築し，その有効性を評価した．学習に利用するメトリクスの組み合わせを変化させた 6 種類のモデルを構築し，それらの精度を比較した．実験の結果，従来より利用されてきたプロダクトメトリクスと，本研究の調査対象であるプロセスメトリクスの両方を利用した推薦モデルが，F 値が 0.827 と最も高い精度を示した．

以降，2. 節では本研究の背景となる，メソッド抽出リファクタリングとリファクタリングの履歴調査と支援手法を紹介する．3. 節でプロセスメトリクスの調査実験について述べ，4. 節でリファクタリング推薦モデルへの適用実験について述べる．続く 5. 節で考察と妥当性の脅威を述べ，6. 節で本研究をまとめる．

```
void printOwing() {
  printBanner();

  //print details
  System.out.println ("name:  " + _name);
  System.out.println ("amount: " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount: " + outstanding);
}
```

図 1　メソッド抽出リファクタリング [1]

2. 背　　景

2.1　メソッド抽出リファクタリング

メソッド抽出リファクタリングとは，既存のメソッドの一部を新規メソッドとして抽出するリファクタリング操作である [1]．長すぎるメソッドや，複数の機能が実装されたメソッドを適切に分割することで，ソースコードの可読性や保守性を向上させる．

図 1 にメソッド抽出リファクタリングの例を示す [1]．図 1 中の上側，printOwing メソッドのコメント文 print details 以下の太文字になっている 2 文が抽出され，新たに printDetails メソッドとして定義されている．抽出元の printOwing メソッドには，抽出された printDetails メソッドの呼び出し文が追加されている．

メソッド抽出リファクタリングは適用される頻度が高い重要なリファクタリングである [4]．また，メソッドはソースコード中に大量に存在するため，それらの中からリファクタリング候補の対象を開発者が適切に選別することは難しい．これらの点から，メソッド抽出の支援が必要である．

2.2　リファクタリングの履歴調査と支援手法

開発者のリファクタリング作業の実態を把握することで、効果的なリファクタリング支援の方法を提案することができる．これまでに実施されたリファクタリングの履歴調査と，提案されている支援手法をいくつか紹介する．

Bavota らは，開発者がソースコードのプロダクトメトリクスやソースコード上に存在する不吉な臭い (Bad Smell) [1] に則ってリファクタリングを実施しているかを明らかにするため，リファクタリング履歴を調査した [7]．3 つのオープンソースソフトウェアの開発で実施された，52 種類のリファクタリングパターンを含む 12,922 件のリファクタリング事例を調査対象としている．調査の結果，開発者は必ずしもプロダクトメトリクスに則って，リファクタリングを実施しているわけではないことがわかった．また，一般的にリファクタリングは，不吉な臭い [1] と呼ばれる，将来的に不具合の原因となりうるコード片が存在するソースコードに対して実施が推薦されている．しかし，不吉な臭いが存在するコード片に対してリファクタリングを実施した事例は，調査対象の全リファクタリング事例の 42%で，そのうちの 7%しか不吉な臭いを除去できていないこ

とが明らかになった．Bavota らは，これらの調査結果より，開発者にとって有益なリファクタリングを推薦するためには，プロダクトメトリクスや不吉な臭いの存在などの成果物に基づいたメトリクスのみを利用するのではなく，プロセスメトリクスのように，開発者の特性を考慮できるようなメトリクスの利用の有用性を示唆している．

後藤らは，メソッド抽出が実施されるメソッドの特徴を調査した [5]．3 つのオープンソースソフトウェアの開発において，メソッド抽出が実施された 1,853 個のメソッドを調査対象としている．調査の結果，メソッド抽出が実施されたメソッドは行数が多く，凝集度が低いことがわかった．また，調査結果に基づき，機械学習を用いてメソッド抽出の候補となるメソッドを開発者へ推薦する手法を提案している [3]．

Tsantalis らは，JDeodorant というリファクタリング支援ツールを開発している [2]．このツールは，ソースコードから不吉な臭い [1] を特定し，そのコード片に適用すべきリファクタリングを開発者に提案する．

これらの手法は，いずれもプロダクトメトリクスを用いてリファクタリングを支援する．プロダクトメトリクスの他に，開発履歴を用いてコード片の特徴を計測するプロセスメトリクスがある．開発プロセスはソースコードの品質に影響を与えることが知られており，品質の良し悪しはコード片をリファクタリングすべきか否かの判断材料になる．そのため，プロセスメトリクスを用いたコード片の特徴付けは，リファクタリング対象箇所の特定に有効であると考えられ，プロダクトメトリクスと併用することで，従来の支援手法の改善が期待できる．しかし，未だプロセスメトリクスを用いたリファクタリング支援手法は提案されておらず，手法確立のための調査も実施されていない．

3. プロセスメトリクスの調査

3.1 調 査 方 法

調査方法の概要を図 2 に示す．調査は次の手順で進める．

step 1　特定のリビジョンを 1 つ選択し，そのリビジョンの全メソッドを得る．

step 2　各メソッドに対して，選択したリビジョン以降でメソッド抽出されたかどうかを検査し，抽出ありメソッドと抽出なしメソッドの 2 群に分ける．

step 3　抽出なしメソッド群から，抽出ありメソッドと同じ数だけランダムにメソッドを選択し，これらと抽出ありメソッド群を合わせて調査対象データセットとする．

step 4　開発履歴より，対象データセットの各メソッドのプロセスメトリクスを計測する．

step 5　リビジョンを変更して step 1 – 4 を繰り返し，各リビジョンで取得したメソッドのプロセスメトリクスを最終的な調査対象とする．

step 6　抽出ありメソッドと抽出なしメソッドの 2 群間でメトリクスに有意な差があるか検証する．

開発履歴の長さがプロセスメトリクスの値を左右するため，特定の 1 つのリビジョン内でのメソッドを比較する．メソッド抽出事例の検出には，藤原らの提案した Kenja [8] を用いる．Kenja は，畑らの提案した細粒度な版管理システムである Historage [9] を利用して，メソッド抽出事例を検出する．Historage は，各クラスに実装されたメソッドごとに開発履歴を記録できる．Kenja は Historage を用いて，リビジョン間での変更情報からメソッド抽出が実施されたと推定されるメソッドの組み合わせを特定し，抽出元のメソッドから削除されたコード片と，抽出先のメソッドの類似度が閾値以上のものを，メソッド抽出事例として検出する．その検出精度は，適合率が 0.96，再現率が 0.86 と高いものである．また，大規模な開発履歴に対しても，高速にリファクタリング検出できるという利点がある．本実験では，比較調査のために多くのメソッド抽出事例が必要となるので，このツールを利用した．

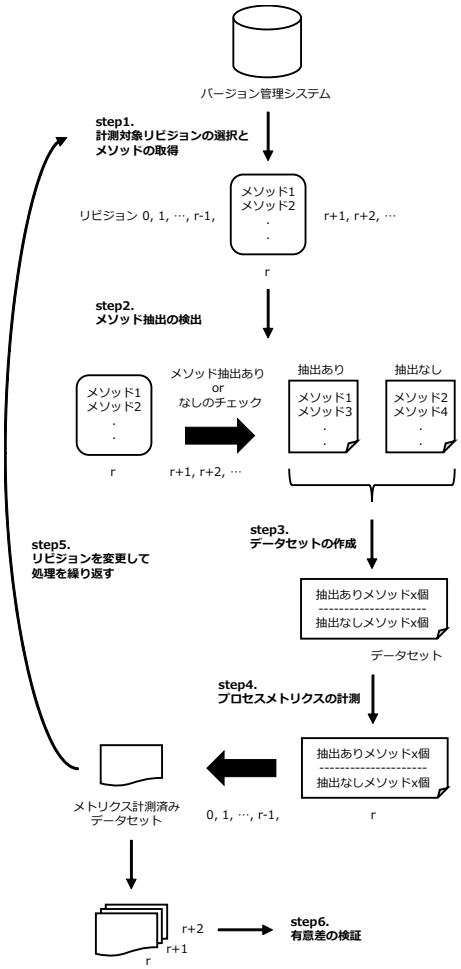

図 2　調査方法の概要

また，選択したリビジョン以降でメソッド抽出されたメソッドを「抽出有りメソッド」とする．これは，メソッド抽出が実施される必要のあるメソッドであっても，即座にメソッド抽出されるわけではないので，選択したリビジョン以降でメソッド抽出された場合，選択リビジョン時点ですでにメソッド抽出の対象となる特徴を持っていると仮定したからである．

計測するメトリクスを表 1 に示す．各メトリクスをクラスレベルとメソッドレベルでそれぞれ計測する．

クラスレベルの計測には，版管理システム Git を利用する．今回の実験では，Java で記述されたソフトウェアを対象とした

表 1　計測プロセスメトリクス一覧

メトリクス	略称
変更回数	NC
オーサー数	NDC
存在期間	AG
ソースコードの追加行数	ADD
ソースコードの削除行数	DEL
ソースコードの変更行数	CHURN

表 2　調査対象ソフトウエア

ソフトウェア	リビジョン数	期間	LOC
JEdit	7,707	2001/09/02 - 2016/02/13	97,116 - 313,706

表 3　計測プロダクトメトリクス一覧

メトリクス	略称
Number of Parameters	PAR
McCabe Cyclomatic Complexity	VG
Nested Block Depth	NBD
Method Lines of Code	MLOC
Weighted methods per Class	WMC
Depth of Inheritance Tree	DIT
Lack of Cohesion of Methods	LCOM
Number of Children	NSC

ため，各クラスは単一のファイルに実装されている．3.1 節の冒頭で説明した方法で選択された特定のリビジョンから過去の開発履歴を辿って，各メトリクスを計測する．

メソッドレベルの計測は，先ほど紹介した Historage [9] を利用する．クラスレベルのメトリクスの計測方法と同様にして，過去の開発履歴を辿り，各メトリクスを計測する．

また，4. 節の実験にて，プロダクトメトリクスを利用した推薦モデルも構築するため，実験対象メソッドのプロダクトメトリクスも併せて計測する．プロダクトメトリクス計測ツールである eclipse metrics plugin(注1)を利用し，表 3 に示すプロダクトメトリクスを計測した．

3.2　対象ソフトウェア

オープンソースプロジェクトである JEdit(注2)を対象に調査した．表 2 に JEdit の情報を示す．開発の進行に応じていくつかのリビジョンを選択し，メトリクス計測対象とした．

今回，調査した JEdit のリビジョン情報を表 4 に示す．リビジョン進行度は，最新リビジョンを進行度 100%とした場合の，最も古いリビジョンからの進行度合いである．様々な開発時期のメソッド抽出事例を調査するため，進行度を 10 – 50%まで変動させたときの各リビジョンを選択した．メソッド数はそのリビジョンに含まれる全メソッドの数を，抽出有，抽出無はそれぞれ，全メソッド内のメソッド抽出対象メソッドと対象ではないメソッドの数を表す．

3.3　結　　果

メソッド抽出対象メソッドと対象ではないメソッドの間で，各メトリクスに有意差があるか確認するため，マン・ホイットニーの U 検定を実施した．表 5 に JEdit の各プロセスメトリクスの中央値，平均値，検定の結果得られた p 値を示す．表 6 にはプロダクトメトリクスの比較結果を示す．p 値が太字で示されたメトリクスは，メソッド抽出有りのメソッドと無しのメソッド間で，有意水準 5%で有意差があったことを示す．

表 4　調査対象リビジョン【JEdit】

リビジョン進行度	コミット日付	メソッド数	抽出有	抽出無
10%	2003/01/11	4,107	5	4,102
20%	2003/03/28	5,080	7	5,073
30%	2005/06/08	3,760	7	3,753
40%	2006/04/22	4,974	14	4,960
50%	2007/02/05	5,649	10	5,639

表 5　プロセスメトリクスの比較結果【JEdit】

メトリクス	レベル	抽出	中央値	平均値	p 値
NC	クラス	有	48.000	122.000	$\boldsymbol{p < 0.05}$
		無	4.000	28.090	
	メソッド	有	2.000	3.093	$0.05 < p$
		無	1.000	2.930	
NDC	クラス	有	1.000	1.884	$\boldsymbol{p < 0.05}$
		無	1.000	1.116	
	メソッド	有	1.000	0.977	$0.05 < p$
		無	1.000	1.000	
AG	クラス	有	118,900,000	98,530,000	$0.05 < p$
		無	73,170,000	93,870,000	
	メソッド	有	86,750,000	89,700,000	$0.05 < p$
		無	73,170,000	87,730,000	
ADD	クラス	有	812	3,256	$\boldsymbol{p < 0.05}$
		無	9	953	
	メソッド	有	5.000	11.350	$\boldsymbol{p < 0.05}$
		無	0.000	4.674	
DEL	クラス	有	951	3,735	$\boldsymbol{p < 0.05}$
		無	23	1,082	
	メソッド	有	1.000	4.628	$0.05 < p$
		無	0.000	4.860	
CHURN	クラス	有	1,690	6,991	$\boldsymbol{p < 0.05}$
		無	60	2,035	
	メソッド	有	5.000	15.980	$\boldsymbol{p < 0.05}$
		無	0.000	9.535	

表 5 より，プロセスメトリクスでは，クラスレベルで NC，NDC，ADD，DEL，CHURN の 5 種類に，メソッドレベルで ADD，CHURN の 2 種類に有意差がみられた．全体的な傾向として，クラスレベルのプロセスメトリクスのほうが，メソッドレベルのプロセスメトリクスよりも値に顕著な差がみられる．そのため，メソッド抽出のようにメソッドを対象としたリファクタリングを推薦する場合は，直接的にメソッドのプロセスメトリクスを利用するのではなく，メソッドが属するクラスレベルのプロセスメトリクスを利用する必要があると考えられる．

表 6 より，プロダクトメトリクスでは，クラスレベルで LCOM，NSC の 2 種類に，メソッドレベルで PAR，VG，NBD，MLOC の 4 種類に有意差がみられた．

（注1）: eclipse metrics plugin: http://metrics.sourceforge.net/

（注2）: JEdit: http://www.jedit.org/

表 6　プロダクトメトリクスの比較結果【JEdit】

メトリクス	レベル	抽出	中央値	平均値	p 値
PAR	メソッド	有	2.000	1.605	$\boldsymbol{p < 0.05}$
		無	1.000	0.875	
VG	メソッド	有	6.000	6.070	$\boldsymbol{p < 0.05}$
		無	2.000	3.906	
NBD	メソッド	有	4.000	3.419	$\boldsymbol{p < 0.05}$
		無	1.000	1.812	
MLOC	メソッド	有	30.000	30.630	$\boldsymbol{p < 0.05}$
		無	3.000	10.120	
WMC	クラス	有	144.000	325.300	$0.05 < p$
		無	57.500	570.000	
DIT	クラス	有	2.000	2.442	$0.05 < p$
		無	2.000	2.438	
LCOM	クラス	有	0.761	0.672	$\boldsymbol{p < 0.05}$
		無	0.694	0.529	
NSC	クラス	有	0.000	0.047	$\boldsymbol{p < 0.05}$
		無	0.000	0.906	

4. 推薦モデルへの適用

メソッド抽出リファクタリングの推薦において，プロセスメトリクスが有効に働くか検証するため，実際に推薦モデルを構築し，その精度を評価する．従来より利用されてきたプロダクトメトリクスも併せて利用し，どのメトリクスが予測に有効に働くか確認する．

4.1 モデル構築方法

ロジスティック回帰モデルを用いて，推薦モデルを構築する．3. 節の実験で得た，開発履歴中でメソッド抽出されたメソッドと，メソッド抽出されなかったメソッドを含むデータセットを用いる．データセット中のメソッドのメトリクスを学習することで構築されたモデルは，あるメソッドが与えられたとき，そのメソッドに対してメソッド抽出を実施するべきか否かを判断する．

実験対象ソフトウェアは JEdit とする．本実験では，モデルの学習に利用するメトリクスの組み合わせを変えた，6 つのモデルの精度を評価する．構築するモデルの一覧を表 7 に示す．ProcessAll は 3. 節の実験で計測した，クラス・メソッドレベルの NC，NDC，AG，ADD，DEL，CHURN の計 12 種類のプロセスメトリクスを学習に利用する．ProcessSignificant は 3. 節の実験で有意差がみられた，計 7 種類のプロセスメトリクスを利用する．同様に，ProductAll は表 3 に示した，計 8 種類のプロダクトメトリクスを利用する．ProductSignificant は 3. 節の実験で有意差がみられた，計 6 種類のプロダクトメトリクスを利用する．All は ProcessAll と ProductAll に用いる計 20 種類のメトリクスを利用し，Significant は ProcessSignificant と ProductSignificant に用いる計 13 種類のメトリクスを利用する．

4.2 結　　果

各推薦モデルを 10 分割交差検定により評価する．評価尺度として，Precision，Recall，F 値を用いる．

Precision は，推薦モデルによってメソッド抽出の対象であるとされたメソッドのうち，実際にメソッド抽出されたメソッドの割合である．Recall は，データセット中に存在する全てのメソッド抽出されたメソッドのうち，モデルによってメソッド抽出の対象であるとされたメソッドの割合である．F 値は，トレードオフの関係にある Precision と Recall を総合的に評価する尺度である．

表 7 に，各推薦モデルの交差検定の結果として得られた Precision，Recall，F 値を示す．太字で示された数値は，各モデル間における Precision，Recall，F 値の最大値である．

Recall は，全てのプロダクトメトリクスを利用して構築した ProductAll が最も高い値を示した．Precision，F 値については，有意差ありのプロセス・プロダクトメトリクスを利用した Significant が最も高い精度を示した．

5. 考　　察

構築した 6 種類のモデルのうち，有意差のあるプロセスメトリクスとプロダクトメトリクスの両方を利用した Significant モデルが最も高い Precision，F 値を示した．F 値が最大である Significant が，今回構築したモデルのなかでは最もバランスの良いモデルであるといえる．また，高い Precision は推薦モデルへの信頼性につながる．Precision の低いリファクタリング推薦モデルは，本来リファクタリングする必要のないコード片を推薦し，ソースコードを改悪させる可能性がある．そのためリファクタリング推薦においては，Recall よりも Precision を重視する必要があると考えられる．これらの点から，メソッド抽出の対象メソッドの推薦においては，プロセスメトリクスとプロダクトメトリクスの両方を利用することが望ましいと考えられる．

実際に，どのメトリクスが予測に有効に働いたのかを調査した．表 8 に最も高い F 値を示した Significant モデルのオッズ比を示す．オッズ比は 1 からの差が大きいほど，その要因の予測に対する影響度が大きいと解釈できる．つまり，表 8 に示した各メトリクスのオッズ比から，モデルがあるメソッドをメソッド抽出するべきか否かを決定する際に，どのメトリクスの値を重要視したかがわかる．なお，正のオッズ比はあるメソッドをメソッド抽出すべきという判断に影響を与え，負のオッズ比はメソッド抽出すべきではないという判断に影響を与えている．表中の太字は，Significant モデルにおいて，正・負のオッズ比が上位 3 つずつのものを示す．また，プロセスメトリクス名には，クラスレベルのものには末尾に_Class を，メソッドレベルのものには同様に_Method をつけている．

正のオッズ比の上位 3 つは LCOM，NBD，ADD_Method であり，負のオッズ比の上位 3 つは，NSC，VG，CHURN_Method である．LCOM，NBD，NSC，VG がプロダクトメトリクスであり，ADD_Method，CHURN_Method がプロセスメトリクスである．従って，正負どちらの予測にも，プロセスメトリクスとプロダクトメトリクスの両方が有効に働いているといえる．

最後に本研究の実験結果の妥当性について述べる．まず，調査対象について述べる．本実験では，開発の進行度に応じた複

表 7　構築したモデル一覧とその精度【JEdit】

モデル	学習メトリクス	precision	recall	F 値
ProcessAll	計測した全てのプロセスメトリクス	0.743	0.486	0.588
ProcessSignificant	有意差ありのプロセスメトリクス	0.778	0.749	0.763
ProductAll	計測した全てのプロダクトメトリクス	0.757	**0.842**	0.797
ProductSignificant	有意差ありのプロダクトメトリクス	0.792	0.808	0.800
All	計測した全てのメトリクス	0.770	0.530	0.628
Significant	有意差ありの全てのメトリクス	**0.825**	0.828	**0.827**

表 8　Significant モデルのオッズ比【JEdit】

メトリクス	オッズ比
NC_Class	0.997
NDC_Class	1.218
ADD_Class	0.999
ADD_Method	**1.871**
DEL_Class	1.001
CHURN_Class	1.000
CHURN_Method	**0.727**
PAR	1.374
VG	**0.553**
NBD	**1.885**
MLOC	1.146
LCOM	**1.893**
NSC	**0.533**

数のリビジョンを選択し，それらからメソッド抽出の対象となるメソッドを取得した．抽出の対象ではないメソッドはランダムに選択し，メソッド抽出の対象となるメソッドと合わせたものを対象データとして実験を行った．そのため，今回選択されなかったリビジョンやメソッドを対象にすると異なる結果が得られる可能性がある．また，対象ソフトウェアはオープンソースソフトウェアの JEdit である．他のソフトウェアに対して実験を行った場合は，今回の調査結果と異なる可能性がある．

次に，リファクタリング検出ツールであるが，本実験では，Kenja を利用してメソッド抽出事例を検出した．Kenja は高い適合率，再現率でメソッド抽出事例を検出するが，誤りや漏れは存在する．そのため，実際にメソッド抽出が行われたメソッドを全て正確に検出し，それらを用いて実験した場合には，今回と異なる結果が得られる可能性がある．

推薦モデルの構築には，ロジスティック回帰モデルを採用している．他のモデル構築方法を利用した場合，本実験の精度とは異なるものが得られる可能性がある．

6. まとめと今後の課題

本研究では，効果的なリファクタリングの支援のために，リファクタリングの実施に相関があると考えられるプロセスメトリクスを用いて，メソッド抽出が実施されたメソッドの特徴を調査した．オープンソースソフトウェアである JEdit から，メソッド抽出の対象となったメソッドと対象とならなかったメソッドを取得し，それらのプロセスメトリクスを計測して，比較した．実験の結果，クラスレベルで NC，NDC，ADD，DEL，CHURN の 5 種類に，メソッドレベルで ADD，CHURN の 2 種類に，メソッド抽出の対象となったメソッドと対象とならなかったメソッド間で有意差がみられた．

また，プロセスメトリクスの調査結果に基づき，JEdit を実験対象として，実際にプロセスメトリクスを利用したメソッド抽出の推薦モデルを構築し，その精度を評価した．推薦モデルに学習させるメトリクスの種類を変えた，計 6 種類のモデルを構築した．全モデルのうち，有意差ありのプロセスメトリクスとプロダクトメトリクスの両方を用いたモデルが最も高い F 値を示した．この結果より，メソッド抽出の推薦には，プロセスメトリクスとプロダクトメトリクスを合わせて用いることが有効であるといえる．

今後の課題として，対象ソフトウェア，計測対象メソッド，プロセスメトリクスの種類を増やした調査の実施が挙げられる．また，プロセスメトリクスは開発履歴の時間的な長さや，リビジョン数などによって値が大きく変わる可能性があるため，それらの条件を考慮した適切な正規化を実施した場合の，リファクタリングとの相関関係を調査する必要があると考えられる．

文　献

[1] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[2] Nikolaos Tsantalis and Alexander Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *The Journal of Systems and Software*, 84:1757–1782, 2011.

[3] 後藤 祥, 吉田 則裕, 藤原 賢二, 崔 恩瀞, and 井上 克郎. 機械学習を用いたメソッド抽出リファクタリングの推薦方法. **情報処理学会論文誌**, 56(2):627–636, 2015.

[4] E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, 38(1):5–18, 2012.

[5] 後藤 祥, 吉田 則裕, 藤原 賢二, 崔 恩瀞, and 井上 克郎. メソッド抽出リファクタリングが行われるメソッドの特徴調査. **コンピュータソフトウェア**, 31(3):318–324, 2014.

[6] 畑 秀明, 水野 修, and 菊野 亨. 不具合予測に関するメトリクスについての研究論文の系統的レビュー. **コンピュータソフトウェア**, 29(1):106–117, 2012.

[7] Gabriele Bavota, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Fabio Palomba. An experimental investigation on the innate relationship between quality and refactoring. *The Journal of Systems and Software*, 107:1–14, 2015.

[8] 藤原 賢二, 吉田 則裕, and 飯田 元. 構文情報リポジトリを用いた細粒度リファクタリング検出手法. **情報処理学会論文誌**, 56(12):2346–2357, 2015.

[9] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Historage: Fine-grained version control system for java. In *Proc. of IWPSE-EVOL '11*, pages 96–100, 2011.