情報処理学会論文誌 Vol.56 No.2 627–636 (Feb. 2015)

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手法

後藤 祥[1] 吉田 則裕[2,a)] 藤原 賢二[3,b)] 崔 恩瀞[1,c)] 井上 克郎[1,d)]

受付日 2014年5月19日, 採録日 2014年11月10日

概要：リファクタリングとは“外部から見たときの振舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理すること”であり，ソフトウェア開発における重要な活動の１つである．本研究では，メソッド抽出というリファクタリングパターンについて，実際にリファクタリングが行われたメソッドを収集し，それらの特徴量を用いた機械学習によって，メソッド抽出リファクタリングの対象を推薦する手法を提案する．実験として，５つのオープンソースソフトウェアに提案手法を適用した結果，メソッド抽出の対象となるメソッドのうち，57%から 96%を特定することができていることが分かった．また，実験の結果から，メソッド抽出が行われるか否かに，メソッドの文の数と凝集度が大きく関与していることが分かった．

キーワード：リファクタリング，機械学習，リポジトリマイニング

Recommending Extract Method Opportunities Using Machine Learnig

Akira Goto[1] Norihiro Yoshida[2,a)] Kenji Fujiwara[3,b)] Eunjong Choi[1,c)] Katsuro Inoue[1,d)]

Received: May 19, 2014, Accepted: November 10, 2014

Abstract: Refactoring is a technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. It is a very important activity to improve software maintainability and readability. In this research, we propose a machine learning approach for suggesting targets of extract method refactoring using source code features. As an experiment to evaluate proposed approach, we applied proposed approach to five open source software. From the result of the experiment, from 60% to 90% of refactored methods can be correctly predicted using the prediction models built by proposed approach. In addition, the number of statements and the degree of cohesion in a single method are strongly related to whether the method is refactored or not.

Keywords: refactoring, machine learning, repository mining

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舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理すること”である [3]．リファクタリングは，主にソフトウェアの保守性や可読性を向上させることを目的としており，ソフトウェア開発における重要な活動の１つである．

大規模なソフトウェアから，開発者が手作業でリファクタリング対象を特定することは困難であるため，リファクタリング対象を自動で特定し開発者へ推薦する手法が提案されており，それらの手法に基づく支援ツールが開発されている [2], [6], [14]．Fowler は，どのようなソースコードに対してリファクタリングを検討すべきかを，ソースコードの Bad Smell として定義しており [3]，リファクタリング対象の推薦はこの Bad Smell を基準にして行われている．

1 大阪大学
Osaka University, Suita, Osaka 565–0871, Japan
2 名古屋大学
Nagoya University, Nagoya, Aichi 464–8601, Japan
3 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
a) yoshida@ertl.jp
b) kenji-f@is.naist.jp
c) ejchoi@ist.osaka-u.ac.jp
d) inoue@ist.osaka-u.ac.jp

© 2015 Information Processing Society of Japan

しかし，Bad Smell には定量的な定義が存在しないことや，プロジェクトごとにリファクタリングを行う基準が異なる場合があることから，リファクタリング対象の推薦基準を決定することは難しい．そのため，既存の支援ツールでは推薦結果のフィルタリングや，実行時のパラメータ設定の変更が可能になっている．しかし，パラメータ設定をどのように変更すればどのように結果が変化するかを開発者が予測することは困難である．

このような問題に対して，実際にリファクタリングが行われた事例を収集し，リファクタリング対象となったソースコードの特徴を用いた機械学習を行うことで，プロジェクトごとのリファクタリングを行う基準を推薦に反映させることができると考えられる．バグ予測などの分野では機械学習を用いた手法が多く提案されており [8], [11], [12]，ソフトウェア開発履歴から抽出した情報を用いたバグ予測に機械学習が有効な手法であることが示されている．

本研究では，メソッド抽出というリファクタリングパターンについて，実際にリファクタリングが行われたメソッドを収集し，それらの特徴量を用いた機械学習によって，メソッド抽出リファクタリングの対象を推薦する手法を提案する．メソッド抽出リファクタリングは，実行される回数の多いことが既存研究 [10] によって明らかにされている点や，メソッドはソフトウェア中に大量に存在し，その中から開発者がリファクタリング対象を選択するのは困難である点から，リファクタリング対象の推薦手法による支援の必要がある．リファクタリング対象の推薦に機械学習を用いる利点としては以下の点があげられる．

- 実際にリファクタリングが行われた対象の特徴を学習することで，パラメータ設定をせずに開発者の考えに合った対象を推薦することができる．
- プロジェクトごとに学習を行うことで，プロジェクトごとにリファクタリングを行う基準が異なる場合でも対応可能である．
- 学習に使用するリファクタリング事例を収集することで，他のリファクタリングパターンへ拡張可能である．

本研究では，機械学習に用いるメソッドの特徴として，メソッドの文の数や凝集度など，21 種類のメトリクスを使用した．これらの特徴をもとに機械学習を用いてメソッド抽出の対象となるかどうかを判別するモデルを作成した．

手法の評価を行うために，オープンソースのソフトウェアに対して適用実験を行った．まず，実験の準備として，実験対象ソフトウェアの履歴にリファクタリング検出ツールを適用し，メソッド抽出が行われたメソッドと行われなかったメソッドを収集した．次に，それぞれのメソッドについて特徴の計測を行い，実験用のデータセットを準備した．適用実験では，これらのデータセットを学習用セットと評価用セットの 2 つに分割して，学習用セットから機械学習で構築した予測モデルを用いることで，評価用セットのメソッドについて，メソッド抽出が行われたかどうかを判別できるかを評価した．実験の結果，提案手法を用いることで，メソッド抽出の対象となるメソッドのうち，57%から 96%を特定することができていることが分かった．また，実験の結果から，メソッド抽出が行われるか否かに，メソッドの文の数と凝集度が大きく関与していることが分かった．

以降，2 章では研究背景について述べる．3 章では，提案手法について説明し，4 章では，適用実験について述べる．最後に，5 章でまとめと今後の課題について述べる．

2. 背景

2.1 メソッド抽出リファクタリング

メソッド抽出はリファクタリングパターンの 1 つであり，既存のメソッドの一部を新たなメソッドとして抽出することによって，メソッドの分割を行う手法である．メソッド抽出の主な目的は，長すぎるメソッドの分割や，複数の機能が実装されたメソッドの分割である．メソッド抽出を行うことの利点としては，メソッドを適切なサイズに分割することで可読性を向上させることができる点や，機能とメソッドを 1 対 1 に対応させることで，機能追加やバグ修正を行う個所を容易に特定できるようになる点などがあげられる．メソッド抽出は，既存の調査研究 [10] において開発者が頻繁に行うリファクタリングの 1 つであることが分かっていることや，他のリファクタリングパターンの一部として行われる場合があることから，適用される頻度が高い重要なリファクタリングである．**図 1** にメソッド抽出の例を示す．図の例では，printOwing メソッドの 5 行目と 6 行目が抽出され，新たに printDetails メソッドとして定義されている．元のメソッドである printOwing メソッドには，printDetails メソッドの呼び出し文が追加されている．

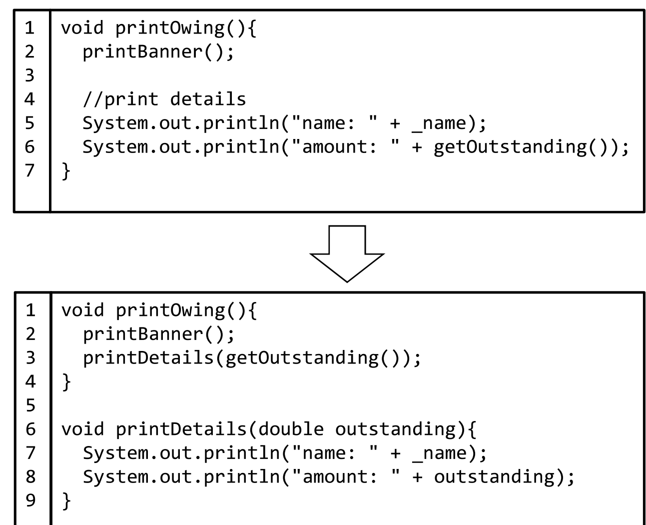


図 1　メソッド抽出リファクタリングの例 [3]
Fig. 1　An example of extract method refactoring.

© 2015 Information Processing Society of Japan

2.2 リファクタリング対象の推薦手法

これまで，リファクタリング対象を特定して開発者に推薦する手法がいくつか提案されている．

Emden らは，コードの不吉な臭い（論文中では Code Smells）を検出，可視化するツールを提案している [2]．Emden らは，コードの不吉な臭いを，対象のソースコードから直接検出できる “Primitive Smell Aspects” と，他の機能から推定することができる “Derived Smell Aspects” の 2 種類に分類し，検出を行っている．

Tsantalis らは，JDeodorant [*1] というリファクタリング支援ツールを統合開発環境 Eclipse のプラグインとして，開発している [14]．JDeodorant は，対象のソースコードから不吉な臭いを検出して開発者へと通知するツールであり，1 つのクラスが巨大である “God Class” や，メソッドが長く複雑である “Long Method” などの複数の不吉な臭いに対応している．

Hotta らは，ソフトウェア中から重複したコードを検出して，Template Method の形成というリファクタリングパターンを適用できる対象を推薦する手法を提案している [6]．Hotta らの手法では，コードクローンの検出結果を用いて，Template Method の形成が適用可能である条件を満たすメソッドの組を探索し，それらのメソッドをリファクタリング対象として開発者へと通知する．

これらの手法では，実行時のパラメータ設定によってリファクタリング対象の推薦基準を変更することや，推薦結果からフィルタリングによって不要なものを取り除くことができる．しかし，パラメータ設定をどのように変更すればどのように結果が変化するかを開発者が予測することは困難である．それに対して本研究では，リファクタリングが行われた事例から学習を行うことによって，プロジェクトごとにリファクタリングを実施する基準が異なっていてもパラメータ設定などを行わずに，その基準に合ったリファクタリング候補を推薦することを目的としている．

2.3 リファクタリング検出ツール

本研究では，機械学習のためにメソッド抽出リファクタリングが行われた事例を収集する必要がある．既存研究では，開発履歴中からリファクタリングが適用された個所を検出する，リファクタリング検出ツールが提案されている．

UMLDiff [16] は 2 つのバージョンから，設計レベルの差分を抽出し，それらの差分をもとに適用されたリファクタリングを検出するツールである．UMLDiff では，まず，入力として与えられた 2 つのバージョンのソースコードに対して，パッケージやクラス，フィールドなどの設計レベルの情報を抽出する．UMLDiff は，32 個のリファクタリングパターンを検出することが可能である．

Ref-Finder [13] は，リファクタリングパターンを論理規則で表現し，それらの規則と 2 つのバージョンから抽出したソースコードレベルの差分を用いて，リファクタリングを検出している．Ref-Finder は，63 個のリファクタリングパターンを検出することができ，現在最も多くのパターンに対応しているリファクタリング検出ツールである．

上記の 2 つのリファクタリング検出ツールは，多くのリファクタリングパターンに対応しており，実験によって高精度で検出可能であることが示されている．その一方で，これらのツールは 2 つのバージョン間でのリファクタリング検出を目的としたものであり，リポジトリの複数のリビジョンから一度に検出を行うには適していない．

本研究では，メソッド抽出事例を収集するために，藤原らが提案している，構文情報を付加したリポジトリを用いるリファクタリング検出ツールを用いる [17]．藤原らのツールは，メソッド抽出とメソッドの引き上げの 2 種類のリファクタリングパターンにしか対応していないが，複数のリビジョンからリファクタリングを高速に検出することができる．本研究では，機械学習のために多くのメソッド抽出事例が必要であったため，藤原らのツールを使用した．藤原らのツールを用いてメソッド抽出リファクタリングの検出を行うと，メソッド抽出の対象となったメソッド，メソッド抽出が行われたコミット，抽出前のコード片と抽出後のメソッド本体とのトークンの類似度などが出力される．この類似度は 0 から 1 の実数値で，高いほどリファクタリングが行われた可能性が高いことを示している

3. 提案手法

提案手法の概要を表した図を**図 2** に示す．提案手法は，ソフトウェアの開発履歴を入力として，メソッド抽出事例の収集，メソッドの特徴量の計測，機械学習を用いたモデル構築の 3 つの処理を行い，あるメソッドがメソッド抽出対象であるかを予測するためのモデルを出力する．以降，それぞれの手順について詳細に説明する．

3.1 Step 1：メソッド抽出事例の収集

機械学習を用いて，メソッド抽出対象を判別するモデルを作るためには，メソッド抽出が行われたメソッドの特徴と，メソッド抽出が行われなかったメソッドの特徴を調べる必要がある．本節では，それぞれのメソッドをどのようにして収集したかを述べる．

まずメソッド抽出が行われたメソッドについて説明する．メソッド抽出事例の収集には，前述した藤原らが提案しているリファクタリング検出ツールを用いる．本研究では，藤原らのツールから出力される類似度の閾値を 0.3 とし，閾値以上の類似度のメソッド抽出事例を使用する．これは，藤原の既存研究において，類似度の閾値が 0.3 のときに最も精度良くメソッド抽出事例の検出が行えることが示

*1 http://www.jdeodorant.com/

© 2015 Information Processing Society of Japan

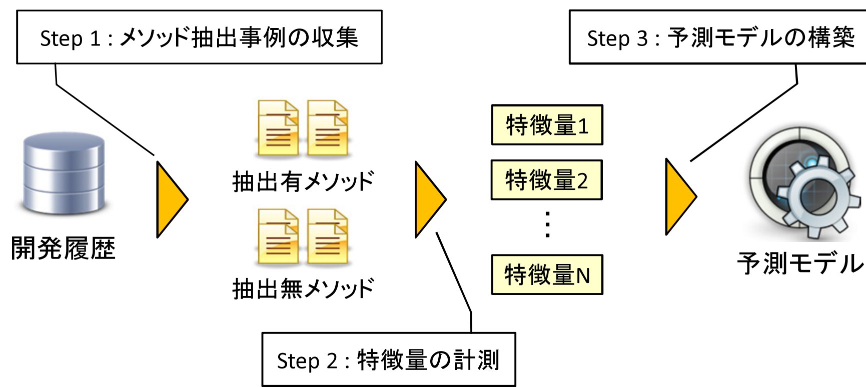


図 2　提案手法の概要図
Fig. 2　An overview of the proposed approach.

表 1　機械学習に使用する特徴量
Table 1　Source code features for machine learning.

カテゴリ	特徴量（略称）
メソッドのサイズ	メソッド中の文の数（NOS）
メソッドのシグネチャ	メソッドの引数の数（ARG），返り値の有無（RET），アクセスレベル（ACCESS）
複雑度 [9]	サイクロマチック数（CYCLO）
凝集度 [15]	Tightness，Coverage，Overlap
メソッドの構文	ループ数（LOOP），if 文数（IF），case 文数（CASE），ブロック数（BLOCK），ネストの深さ（NEST）
メソッド内の変数	ローカル変数の数（VAR）
CK メトリクス [1]	WMC，DIT，CBO，NOC，RFC，LCOM
コードクローン	コードクローンの有無（CLONE）

されているためである [17]．閾値でフィルタリングを行った後，メソッド抽出の対象となったメソッドについて，抽出が行われる直前のリビジョンのものを取得する．これらのメソッドをメソッド抽出が行われたメソッドとして，以降の特徴量の計測や機械学習に用いる．

次に，メソッド抽出が行われなかったメソッドの収集方法について説明する．本研究では，メソッド抽出が行われなかったメソッドは，リポジトリからランダムに収集する．具体的な手順は，下記のとおり．

(1) 全リビジョンの中からランダムにリビジョンを 1 つ選択する．
(2) 選択したリビジョンのソースコードに含まれる全メソッドを列挙する．
(3) 列挙したメソッドの集合から，次のコミットでメソッド抽出が行われていないメソッドの 1 つをランダムに選択する[*2]．

この手順を，学習に必要なメソッドが集まるまで繰り返すことで，メソッド抽出が行われなかったメソッドを収集する．

[*2] 次のコミット以降において，メソッド抽出が行われる可能性があるが，ランダムに選択したリビジョンにおけるソースコードの特徴が直接影響している可能性が高い直後のコミットのみについて，メソッド抽出の有無を確認した．

3.2　Step 2：特徴量の計測

Step 1 で収集したメソッドに対して，特徴量の計測を行う．**表 1** に本研究で使用する 21 種類の特徴量を示す．このうち，CK メトリクス（6 種類）については収集したメソッドを含むクラスに対して計測し，それ以外のメトリクス（15 種類）については収集したメソッドに対して計測する．リファクタリングは可読性や保守性の向上を目的としており，それらが低いソースコードに対して行われていると考えられる．そのため，本研究では，可読性や保守性に関連する特徴量を選択した．

クローンの有無については，コードクローン検出ツールである CCFinder [7] を用いて検出する．その他の特徴量については，ソースコード解析ツール MASU [5] や Eclipse JDT [*3] を用いて計測する．

本研究では，凝集度を表す特徴量として，メソッドレベルの凝集度メトリクスであるスライスベースの凝集度メトリクスを用いる [15]．スライスベースのメトリクスの計算に必要となるメソッドの出力変数は，Tsantalis らの定義に従い，メソッドの返り値と，メソッドの本体とスコープが一致する変数とした [14]．スライスベースのメトリクスは，メソッドの出力変数に基づいて凝集度を計算するものであり，プログラムスライシングの基準となる変数が存在しないメソッドに対しては，凝集度を計算することができ

[*3] http://www.eclipse.org/jdt/

© 2015 Information Processing Society of Japan

表 2 実験対象ソフトウェア
Table 2 Target projects for the case study.

ソフトウェア名	リビジョン数	ファイル数	メソッド数
Ant	12,783	1,222	12,369
ArgoUML	17,748	1,904	14,284
jEdit	5,787	576	6,632
jFreeChart	916	1,060	10,495
Mylyn	8,414	1,028	8,936

ない．**表 2** の全メソッドのうち，40%についてはスライスベースのメトリクスを計算できなかった．

3.3 Step 3：予測モデルの構築

計測した特徴量を基に，機械学習アルゴリズムを用いて，メソッド抽出の対象を判別するモデルを構築する．機械学習には，フリーのデータマイニングツールである Weka [*4]を用いる．Weka は，機械学習アルゴリズムだけでなく，変数選択アルゴリズムや予測モデルの評価機能など，機械学習に関連した多くの機能を実装したツールである．

予測モデルを構築する前に，データセット中の欠損値の補完と，変数選択を行う．データセット中の欠損値の補完は，出力変数が存在しないメソッドに対して，値を計算することができないスライスベースのメトリクス Tightness，Coverage，Overlap に対する処理である．なお，3.2 節で述べたとおり，表 2 の全メソッドのうち 40%についてはこれらメトリクスを計算できなかった．欠損値が存在すると正しく予測することができないため，欠損値が存在する要素を除外するか，定数値やその特徴の平均値などで欠損値を補完する必要がある．本手法では欠損値の補完法の 1 つである，その特徴量の平均値を用いた補完を行う．欠損値の補完のあと，有用な特徴量のみをモデルの構築に用いるために，変数選択を行う．変数選択アルゴリズムは複数提案されており，対象データセットや使用する予測モデルによってどのアルゴリズムが優れているかは異なる．Hall らが行った変数選択アルゴリズムの比較 [4] によると，ラッパー法を用いて変数選択を行うと多くの場合予測性能が最も良いという結果が示されている．一方でラッパー法は，変数間に強い相互作用がある場合は他の手法と比較して予測性能が落ちるという問題がある [4]．しかし，本研究で用いる特徴量において，ラッパー方の性能を大きく低下させるような強い相互作用があるかどうか確認されていないため，Hall らのベンチマーク [4] において多くの場合に優れた予測性能を示したラッパー法を本研究では採用した．

提案手法では，収集したメソッド抽出事例とその特徴量を学習データとして，決定木とロジスティック回帰，ベイジアンネットの 3 種類の予測モデルを構築する．構築した予測モデルでは，あるメソッドとその特徴量が与えられたときに，そのメソッドに対してメソッド抽出を行うべきであるか判断することができる．そのため，メソッド抽出対象の推薦にあたっては，まず対象のメソッドの特徴量の計測を行い，次にそれらの特徴量を予測モデルに与えてメソッド抽出の対象であるか判別し，開発者に通知するという手順になる．

[*4] http://www.cs.waikato.ac.nz/ml/weka/

4. 適用実験

提案手法の有効性を評価するために，オープンソースのソフトウェアに対して提案手法を適用した．本研究では，実験対象として 5 個のオープンソースソフトウェアを選択した．実験対象の一覧を表 2 に示す．これらのソフトウェアはすべて Java 言語を用いて記述されている．また，表 2 中のファイル数とメソッド数は，最新リビジョンにおける Java のファイル数とメソッド数である．本章では，実験の手順と評価尺度について述べる．

4.1 実験手順

まず，表 2 に示したソフトウェアに対して，藤原らのリファクタリング検出ツールを適用して，メソッド抽出が行われた事例の収集を行う．そして，メソッド抽出が行われなかったメソッドをメソッド抽出が行われたメソッドと同じ数だけランダムに選択し，これらを合わせて実験用データセットとする．

次に，実験用データセットのメソッドの特徴量の計測，モデルの構築と評価を行う．モデルの構築と評価には，10 分割交差検定を用いる．10 分割交差検定は，データセットを 10 個のブロックに分割し，そのうち 1 個を評価セット，残り 9 個を学習セットとしてモデルの構築と評価を行うという処理を，評価セットに用いるブロックを変化させながら 10 回繰り返す手法である．本実験では，モデルの構築に用いる学習セットはメソッド抽出が行われたメソッドと行われなかったメソッドの割合は 50%ずつのものを用いて，評価セットには，メソッド抽出が行われたメソッドの割合を 50%から 10%まで 10%刻みで変化させて評価を行う．これは，評価セット中のメソッド抽出が行われたメソッドの割合を変化させたときに，どのように結果が変化するかを確認するためである．

4.2 評価尺度

モデルの評価には，Precision と Recall，F 値という 3 つの評価尺度を用いる．以下それぞれの尺度の定義について説明する．

Precision は，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が行わ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，すべてのメソッド抽出が行わ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

© 2015 Information Processing Society of Japan

ソッドの割合である．そして，トレードオフの関係にある Precision と Recall を総合的に評価するため，これらの値の調和平均である F 値を使用する．

以下に，Precision と Recall，F 値の定義式を示す．式において，E をメソッド抽出が行われたメソッドの集合，C_E をモデルによってメソッド抽出対象であるとされたメソッドの集合とする．

$$Precision = \frac{|E \cap C_E|}{|C_E|} \quad (1)$$

$$Recall = \frac{|E \cap C_E|}{|E|} \quad (2)$$

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (3)$$

表 3　メソッド抽出リファクタリングの検出結果

Table 3　Result of extract method refactoring detection.

	メソッド抽出数	データセットのメソッド数
Ant	766	1,532
ArgoUML	740	1,480
jEdit	502	1,004
jFreeChart	90	180
Mylyn	490	980

4.3　実験結果

表 3 に，藤原らのリファクタリング検出ツールによって，検出されたメソッド抽出事例の数を示す．jFreeChart のみ，他のソフトウェアに比べリビジョン数が少なかったため，検出されたメソッド抽出の事例数も少なくなっている．jFreeChart 以外のソフトウェアでは，490 個から 766 個のメソッド抽出事例数が検出された．次に各ソフトウェアごとに，メソッド抽出が行われなかったメソッドを，メソッド抽出が行われたメソッドと同じ数だけランダムに選択した．表 3 に，作成したデータセット中のメソッド数を示す．

次に，データセットを用いて予測モデルの構築と評価を行った結果を示す．Precision と Recall，F 値について，結果を表したグラフを**図 3** に示す．グラフの縦軸は各評価値，横軸は評価セット中のメソッド抽出が行われたメソッドの割合である．これらの図には，各ソフトウェアに対する結果と比較のためのベースラインの結果（図中の Base）を示している．ベースラインの結果とは，あるメソッドが与えられた際に，メソッド抽出の対象であるかどうかをランダム（50%ずつの確率）で返すモデルに対する結果である．このようなランダムなモデルを用いた結果よりも良い

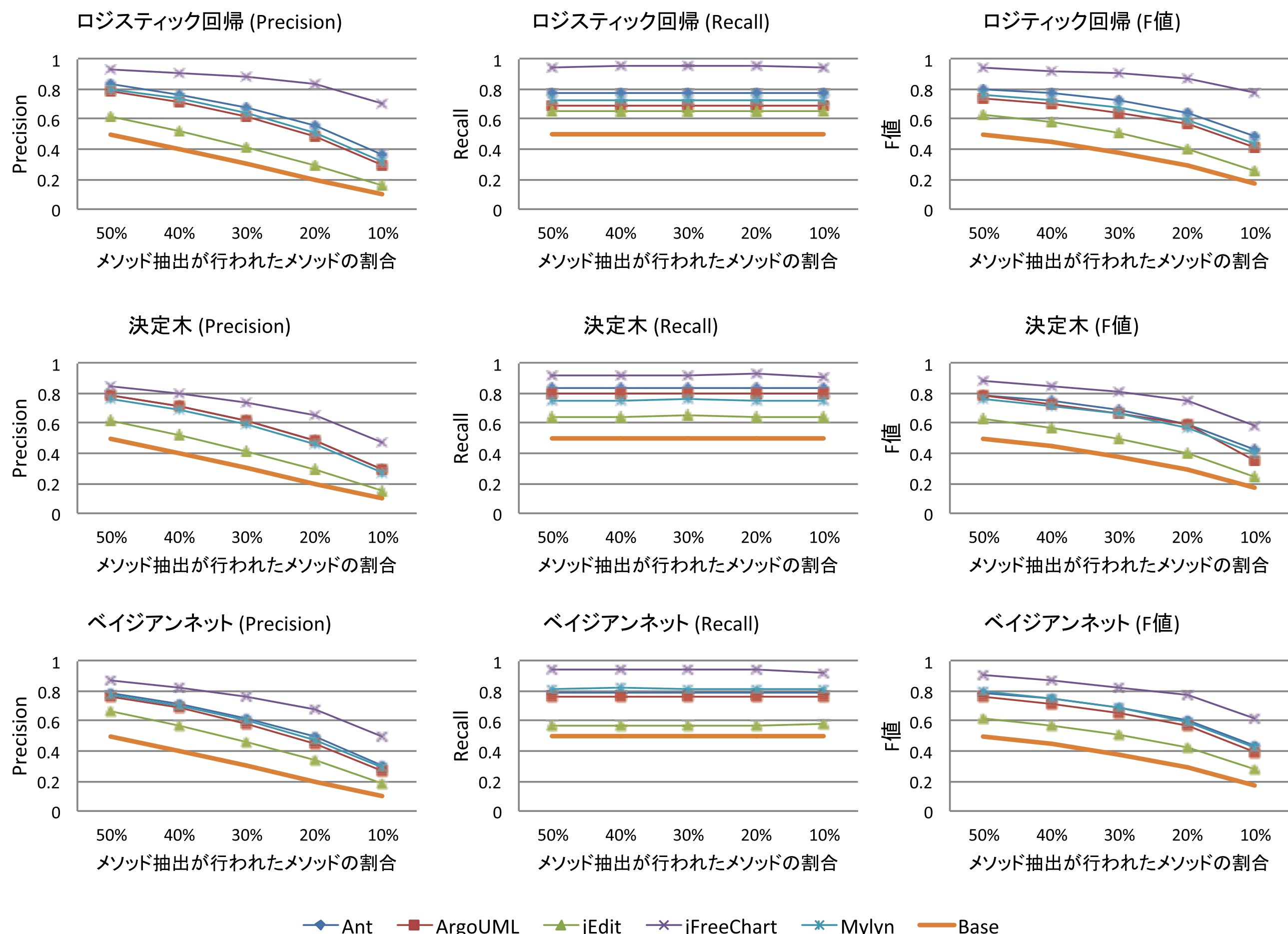


図 3　評価結果（Precision，Recall，F 値）

Fig. 3　Evaluation results (Precision, Recall, F-score).

© 2015 Information Processing Society of Japan

情報処理学会論文誌 Vol.56 No.2 627–636 (Feb. 2015)

表 4 各特徴量が選択された回数
Table 4 The number of selections for each feature.

	Ant	ArgoUML	jEdit	jFreeChart	Mylyn	合計
NOS	2	3	2	3	3	13
Coverage	3	2	3	2	3	13
ARG	3	2	3	3	1	12
BLOCK	3	3	1	3	2	12
CBO	3	3	1	3	2	12
RET	3	3	2	2	1	11
IF	3	1	2	3	2	11
WMC	1	2	2	3	3	11
DIT	2	2	2	3	2	11
ACCESS	3	2	3	1	1	10
LOOP	2	3	2	1	2	10
NEST	1	1	2	3	3	10
RFC	2	3	1	3	1	10
Tightness	1	3	2	1	2	9
VAR	3	2	1	1	1	8
NOC	2	2	1	2	1	8
LCOM	2	1	3	1	1	8
CYCLO	1	1	1	2	2	7
CLONE	0	2	1	1	3	7
Overlap	1	1	1	2	1	6
CASE	1	0	2	1	2	6

結果であれば，使用した特徴量がメソッド抽出が行われるかどうかに関係したものであり，学習の効果があったことを意味する．

グラフをみると，すべての場合においてベースラインを上回る結果となっていることが分かる．それぞれの評価値ごとにみると，Precision は評価セットの割合によって値が大きく変化しているが，Recall は，ほぼ横ばいで値に変化がみられなかった．Recall については，ほとんどの場合においてが 0.6 から 0.9 程度の値となっており，メソッド抽出の対象であるメソッドのうち，57%から 96%が提案手法によって特定できていることが分かった．モデルごとの結果をみると，使用するモデルによって大きな差がないが，それぞれのモデルごとに結果の平均値を調べた結果，Precision についてはロジスティック回帰が優れており，Recall については決定木が優れていることが分かった．対象ソフトウェアごとに違いをみると，jFreeChart に対する結果が良く，jEdit に対する結果が悪くなっている．その他の 3 つのソフトウェアについては，評価値に大きな違いはみられなかった．

次に，変数選択の結果から，どの特徴量が予測に有用であったかを調べた結果を示す．**表 4** は，変数選択によって各特徴量が選択された回数を示している．表 4 では，各ソフトウェアごとの特徴量が選択された回数と，合計回数を示しており，合計回数で降順に並べている．本手法で変数選択に用いているラッパー法は，各モデルごとに変数選択を行う手法である．そのため，各ソフトウェアについて 3 回変数選択を行っており，各ソフトウェアにおける選択回数の最大値は 3，合計の選択回数の最大値は 15 となる．

表 5 メソッド抽出リファクタリングを行った開発者数
Table 5 The numbers of developers who performed extract method refactoring.

	開発者数
Ant	23
ArgoUML	24
jEdit	14
jFreeChart	1
Mylyn	14

表 4 の結果から，本実験においてはメソッドの文の数である NOS と，スライスベースの凝集度メトリクスである Coverage が最も選択された特徴量であることが分かった．各ソフトウェアにおける結果を比較すると，対象ソフトウェアによって有用な特徴量が異なることが分かる．たとえば，ブロック数を表す BLOCK は，Ant と ArgoUML，jFreeChart ではすべてのモデルにおいて選択されているが，jEdit では 1 度しか選択されていない．

4.4 考察

まず，各ソフトウェアごとの結果について考察する．各ソフトウェアごとの結果の差をみると，jFreeChart に対する結果が良く，jEdit に対する結果が他のソフトウェアに比べて悪かった．jFreeChart についてより詳細に開発履歴の調査を行った結果，検出されたメソッド抽出事例が同一の開発者によって行われていたことが分かった（**表 5**）．このことから，jFreeChart ではメソッド抽出が行われるメソッドの特徴が一貫しており，予測に有効に作用したのではないかと考えられる．jEdit については，データセットの特徴量を調べた結果，メソッド抽出が行われたメソッドと行われなかったメソッドで，特徴量の値に大きな差がないことが分かった．そのため，jEdit についてはモデルによる予測性能が低かったと考えられる．このように，予測性能には使用するデータセットが大きく影響することが分かった．

次に，評価セット中のメソッド抽出が行われたメソッドの割合を変化させた場合の，結果の変化について考察する．図 3 に示したグラフから，メソッド抽出が行われたメソッドの割合を減少させても，本実験においてベースラインとしたランダムに予測を行うモデルより良い結果であることが分かる．これは，今回選択した特徴量がメソッド抽出が行われるかどうかに関連しており，学習によって予測性能が向上したことを示している．評価値ごとの変化をみると，Recall に大きな変化はないが，Precision はメソッドの

© 2015 Information Processing Society of Japan

割合の減少にともなって低下していることが分かる．そのため，メソッド抽出が行われたメソッドの割合を減少させても高い予測性能を維持できるように，手法を改善する必要があると考えられる．手法の改善案としては，特徴量の追加，出力結果のフィルタリングやデータセットを開発者ごとに分割するなどの方法があげられる．データセットを開発者ごとに分割する方法は，jFreeChart に対する予測性能が良かったことから，有効な手段であると考えられる．

図 3 から，最も Precision が低い傾向にある jEdit であっても，メソッド抽出が行われたメソッドの割合が 50%のとき，Precision が 0.6 以上になっている．jEdit の場合，データセット中の 1,004 個のメソッドを，10 分割交差検定のために 904 個の学習セットと 100 個の評価セットに分割している．メソッド抽出が行われたメソッドの割合が 50%のとき，904 個の学習セットの半数の 452 個がメソッド抽出事例である．これらのことから，Precision を 0.6 以上に保つためには 450 程度のメソッド抽出事例を学習セットに含めることが望ましいと考えられる．表 2 および表 3 より，1 メソッド抽出事例あたりのリビジョン数は 10.2～24.0 であるため，450 の事例を収集するためには，4,600～10,000 リビジョンが必要であると考えられる．開発プロジェクトの早期段階には，このようなリビジョン数の履歴が存在することは稀であると考えられるため，類似した成熟プロジェクトのメソッド抽出事例を流用する方法の考案をすべきと考えられる．

各特徴量の有用性について述べる．表 4 より，メソッドの文の数である NOS と，スライスベースの凝集度メトリクスである Coverage が最も選択された特徴量であった．NOS は，メソッドのサイズを表す特徴量であり，この特徴量が多く選択されていたことから，メソッド抽出を行うかどうかの判断において，メソッドのサイズが重要な要因であることが分かる．Coverage は，メソッドの内の出力変数に関連した文の数とメソッドの文数の比の平均を表している．Coverage が低いことは，出力変数に関連のない文がメソッドに多く存在することを意味している．このことから，メソッドの出力変数に関連した文とそうでない文をメソッド抽出を用いて別々のメソッドに分割しているのではないかと考えられる．

変数選択によって選択された回数が最も少ない特徴量は，スライスベースの凝集度メトリクスである Overlap と，メソッド中の case 文の数を表す CASE であった．これらの 2 つの特徴量について調査したところ，ともに値の分布が偏っていることが分かった．Overlap は，メソッド内に出力変数が 1 つしかない場合は 1 になり，スライスがまったく重複しない出力変数が存在する場合は 0 になるため，多くのメソッドの Overlap の値は 0 か 1 となる．また CASE についても，今回の対象のメソッドの多くは case 文が使われていなかった．

変数選択の結果，多く選択される特徴量とあまり選択されない特徴量が分かったが，最も選択されなかった特徴量でも 15 回のうち 6 回選択されている．また，各ソフトウェアにおける変数選択の結果も異なるものであり，どの特徴量が有用であるかは変数選択アルゴリズムを適用するまで分からない．そのため，本提案手法のように，特徴量を計測する段階では多くの特徴量を計測して，モデルの構築を行う際に変数選択によって有用なものを選別する方法が有効であると考えられる．

4.5　妥当性への脅威

本節では，本実験における妥当性への脅威について述べる．

まず，使用したリファクタリング検出ツールについてである．本手法では，藤原らのリファクタリング検出ツールが出力した結果のうち類似度の閾値が 0.3 以上のものについて，メソッド抽出が行われたと判断して利用している．そのため，藤原らのツールの出力結果に誤検出が多く含まれている場合や，開発履歴から検出ができなかったメソッド抽出事例が多く存在すると，実験結果に悪影響を与える可能性がある．ただし，藤原らがツールの検出結果を目視で確認した結果，Precision が 0.96，Recall が 0.86 と，他のリファクタリング検出ツールと比べて高い精度であることが示されており [17]，藤原らのツールの検出結果による本実験への大きな影響はないと考えられる．

メソッド抽出されなかったメソッドの収集の際，直後のコミットにおいてメソッド抽出されなかったもののみを収集したが，直後のコミット以降にメソッド抽出される可能性が考えられる．本研究では，ソースコードの特徴が直接影響している可能性が高い直後のコミットのみにおいて，メソッド抽出の有無を判定した．直後のコミット以降においてメソッド抽出されたメソッドについても，メソッド抽出されたメソッドとして扱うと，実験結果が変化する可能性がある．

本研究では，実験対象として 5 つのオープンソースのソフトウェアを使用したが，今回の実験結果が対象ソフトウェアに限定されたものである可能性がある．特に，本実験で最も良い結果となった jFreeChart については，他の対象ソフトウェアに比べてリビジョン数が少ない，検出されたメソッド抽出事例数も少なかった．そのため，今後様々な規模のソフトウェアに対して実験を行い，結果がどのように変化するか確認する必要がある．

本研究では，メソッド抽出が行われるかどうかを判別するための特徴量として，メソッドを対象とした特徴量を 15 種類，クラスを対象にした特徴量を 6 種類用いた．本研究では，クラスやメソッドの特徴を定量化し，かつ可読性や保守性と関連するメトリクスを特徴量として実験に用いたが，今後既存リファクタリング候補の推薦ツールの出力

© 2015 Information Processing Society of Japan

を特徴量として取り入れる方法が考えられる．たとえば，JDeodorant [14] を用いて，メソッド抽出すべきと判定されるかどうかの真偽値を特徴量として取り入れるという方法が考えられる．しかし，リファクタリング候補の推薦ツールは，入力する設定値で出力が変化するため，特徴量を計測する際に設定値を指定する方法について考案する必要がある．

5. まとめと今後の課題

本研究では，機械学習を用いてメソッド抽出リファクタリングの対象を推薦する手法を提案した．提案手法では，メソッド抽出の対象となったメソッドについて，サイズや複雑度など 21 種類の特徴量を計測し，それらを用いて推薦のための予測モデルを構築した．

実験として，5 つのオープンソースソフトウェアに対して手法を適用し評価を行った．評価の結果，メソッド抽出対象となるメソッドのうち 57%から 96%を提案手法によって特定できていることが分かった．また，変数選択アルゴリズムの適用結果から，メソッド中の文の数と，スライスベースの凝集度メトリクスである Coverage が有用な特徴量であることが分かった．

今後の課題としては，まず予測性能の向上があげられる．これについては，特徴量の追加やデータセットを開発者ごとに分割するなどの方法によって改善することができると考えられる．次に，ソフトウェア開発プロジェクトの早期段階に提案手法を適用する方法を考案する必要がある．たとえば，早期段階のプロジェクトに類似した成熟プロジェクトがあれば，成熟プロジェクトにおいて収集したメソッド抽出事例を基に予測モデルを構築し，早期段階のプロジェクトにおけるメソッド抽出の推薦に活用するという方法が考えられる．このためには，プロジェクト間の類似性を判定する方法を考案する必要がある．予測モデルを流用する際に変更を加える必要性の判定方法や，必要性があると判定された場合の変更方法も研究課題になると考えられる．最後に，成熟プロジェクトにおいて提案手法を適用する場合，早期段階のメソッド抽出事例も予測モデルに取り入れるべきかについて調査する必要がある．早期段階のメソッド抽出事例が予測モデルに悪影響を及ぼしている場合は，どの程度新しい事例であれば予測モデルに取り入れるべきかについて調査する必要があると考えられる．

謝辞 本研究は JSPS 科研費 26730036，25220003 の助成を得たものである．

参考文献

[1] Chidamber, S.R. and Kemerer, C.: A metrics suite for object oriented design, *IEEE Trans. Softw. Eng.*, Vol.20, No.6, pp.476–493 (1994).

[2] Emden, E.V. and Moonen, L.: Java quality assurance by detecting code smells, *Proc. WCRE*, pp.97–106 (2002).

[3] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).

[4] Hall, M.A. and Holmes, G.: Benchmarking attribute selection techniques for discrete class data mining, *IEEE Trans. Knowl. Data Eng.*, Vol.15, No.6, pp.1437–1447 (2003).

[5] Higo, Y., Saitoh, A., Yamada, G., Miyake, T., Kusumoto, S. and Inoue, K.: A pluggable tool for measuring software metrics from source code, *Proc. IWSM-MENSURA*, pp.3–12 (2011).

[6] Hotta, K., Higo, Y. and Kusumoto, S.: Identifying, tailoring, and suggesting form template method refactoring opportunities with program dependence graph, *Proc. CSMR*, pp.53–62 (2012).

[7] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol.28, No.7, pp.654–670 (2002).

[8] Lee, T., Nam, J., Han, D., Kim, S. and Hoh, I.P.: Micro interaction metrics for defect prediction, *Proc. ESEC/FSE*, pp.311–321 (2011).

[9] McCabe, T.J.: A complexity measure, *IEEE Trans. Softw. Eng.*, Vol.2, No.4, pp.308–320 (1976).

[10] Murphy-Hill, E., Parnin, C. and Black, A.P.: How we refactor, and how we know it, *IEEE Trans. Softw. Eng.*, Vol.38, No.1, pp.5–18 (2011).

[11] Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K. and Murphy, B.: Change bursts as defect predictors, *Proc. ISSRE*, pp.309–318 (2010).

[12] Nam, J., Pan, S.J. and Kim, S.: Transfer defect learning, *Proc. ICSE*, pp.382–391 (2013).

[13] Prete, K., Rachatasumrit, N., Sudan, N. and Kim, M.: Template-based reconstruction of complex refactorings, *Proc. ICSM*, pp.1–10 (2010).

[14] Tsantalis, N. and Chatzigeorgiou, A.: Identification of extract method refactoring opportunities for the decomposition of methods, *Journal of Systems and Software*, Vol.84, No.10, pp.1757–1782 (2011).

[15] Weiser, M.: Program slicing, *Proc. ICSE*, pp.439–449 (1981).

[16] Xing, Z. and Stroulia, E.: Refactoring detection based on UMLDiff change-facts queries, *Proc. WCRE*, pp.263–274 (2006).

[17] 藤原賢二，吉田則裕，飯田　元：ソフトウェアリポジトリを対象とした細粒度リファクタリング検出，ソフトウェア工学の基礎ワークショップ，pp.101–106 (2013).

後藤 祥

平成 24 年大阪大学基礎工学部情報科学科卒業．平成 26 年大阪大学院情報科学研究科博士前期課程修了．現在，新日鉄住金ソリューションズ株式会社に勤務．在学中，リファクタリング支援に関する研究に従事．

© 2015 Information Processing Society of Japan

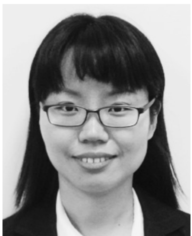
情報処理学会論文誌　Vol.56　No.2　627–636 (Feb. 2015)

吉田 則裕　（正会員）

平成 16 年九州工業大学情報工学部知能情報工学科卒業．平成 21 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．平成 26 年より名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

藤原 賢二　（学生会員）

平成 22 年大阪府立工業高等専門学校総合工学システム専攻修了．平成 24 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，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 3 年．リファクタリングの適用履歴分析，プログラミング教育支援に関する研究に従事．



崔 恩瀞　（学生会員）

平成 24 年大阪大学大学院情報科学研究科博士前期課程修了．現在，大阪大学大学院情報科学研究科博士後期課程 3 年．コードクローン管理やリファクタリング支援手法に関する研究に従事．

井上 克郎　（フェロー）

昭和 59 年大阪大学大学院基礎工学研究科博士後期課程修了（工学博士）．同年大阪大学基礎工学部情報工学科助手．昭和 59～61 年ハワイ大学マノア校コンピュータサイエンス学科助教授．平成 3 年大阪大学基礎工学部助教授．平成 7 年同学部教授．平成 14 年大阪大学大学院情報科学研究科教授．平成 23 年 8 月より大阪大学大学院情報科学研究科研究科長．ソフトウェア工学，特にコードクローンやコード検索等のプログラム分析や再利用技術の研究に従事．

© 2015 Information Processing Society of Japan