情報処理学会論文誌 Vol.56 No.2 692–700 (Feb. 2015)

API呼び出しとメソッド周辺の識別子の実績に基づいたAPI集合推薦手法

早瀬 康裕[1,a)] 鬼塚 勇弥[2] 山本 哲男[3,b)] 石尾 隆[2,c)] 井上 克郎[2,d)]

受付日 2014年5月19日, 採録日 2014年11月10日

概要：現代の多くのソフトウェア開発では，膨大なライブラリから必要なAPIを選び出し，それをどのように組み合わせればよいかを調べなければならない．本研究では，開発者が新規作成したいメソッドの名前を記述したときに，そのメソッド本体で使用される可能性のあるAPIの集合を推薦することで，APIの選択を支援する手法を提案する．使用される可能性のあるAPIの推薦には大規模なソースコード集合から抽出した相関ルールを使用する．相関ルールには，メソッド名などの識別子から得た情報と，使用されたAPIとの関係が記録されている．本手法によって開発者に有用なAPI集合を推薦できるかを調査したところ，推薦の上位に適切なAPIが含まれていることを確認した一方，手法に改善の余地があることも分かった．

キーワード：API，メソッド本体，相関ルールマイニング，コード補完

Recommending API Sets for a New Method Based on Surrounding Identifiers and API Usage History

Yasuhiro Hayase[1,a)] Yuya Onizuka[2] Tetsuo Yamamoto[3,b)]
Takashi Ishio[2,c)] Katsuro Inoue[2,d)]

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

Abstract: In modern software development, developers have to select and combine appropriate APIs from vast amount of software libraries to implement features. This paper proposes an approach that takes as input a method name which a developer is attempting to create, and suggests APIs that are likely used as a template of method body. By using the template as a reference and/or editing the template, the developer can write the method body. Our approach generates templates from association rules that associate APIs with identifiers such as method names, class names, and field names included in a large set of source files.

Keywords: API, method body, association rule mining, code completion

1. まえがき

高品質なソフトウェアを短期間で開発することが社会から求められており，それに応えるための方法の1つとして過去に開発されたソフトウェアの再利用が行われている．社会的要求の背景には，社会基盤がソフトウェアへ依存する度合いが高まっていることや，急速な技術の進歩に遅れをとらないために納期が短くなっていることなどがある．過去に開発されたソフトウェアは新規に開発するソフトウェアよりも品質が高いことが多いため，ソフトウェア

1 筑波大学システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba, Iraraki 305–8573, Japan
2 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
3 日本大学工学部情報工学科
Department of Computer Science, College of Engineering, Nihon University, Koriyama, Fukushima 963–8642, Japan
a) hayase@cs.tsukuba.ac.jp
b) tetsuo@cs.ce.nihon-u.ac.jp
c) ishio@ist.osaka-u.ac.jp
d) inoue@ist.osaka-u.ac.jp

の再利用は開発期間の短縮とソフトウェアの品質とを両立させる方法だと考えられている．ソフトウェアの再利用には，目的に適った再利用対象ソフトウェアを発見し，それを利用する適切な方法を調べる必要があるが，この作業に開発者の手間と時間とを要するという難しさがある．

ソフトウェア再利用の方法の 1 つに，Application Programming Interface（API）を通して既存ソフトウェアを利用する方法がある．多種多様な API を組み合わせて目的の機能を実装することは，現代的なソフトウェア開発の一般的な手順となっている．たとえば多くのウェブアプリケーションでは，ウェブアプリケーションフレームワークに加えて，複数のライブラリ（例：クライアントから送られるデータを処理するライブラリや，データベースにアクセスするライブラリなど）を組み合わせて利用する．このように，ソフトウェア開発者は適切なライブラリとそこで提供されている API を選択し，適切な組合せを発見するという手順を日常的に繰り返している．

現在では多種多様なライブラリが提供されているため，多くの機能が再利用によって実現できるようになっているが，その一方で適切なライブラリと API を組み合わせてソフトウェア開発を行うのは，難しく手間がかかる作業となっている．なぜならば，開発者は目的を達成するために必要なすべての機能を調べ出し，その機能を提供するライブラリを見つけ，そのライブラリの API を組み合わせて利用する方法を調べなければならないためである．さらに，ライブラリには同時に使用することのできない組合せが存在するため，個々の機能を提供するライブラリを見つけたとしても，目的を達成できるとは限らない．

ライブラリや API を利用することの難しさを軽減することを目的として，開発者がこれから書こうとしているコードを，直前に書かれた文の並びから予測して推薦する手法が提案されている．Bunch ら [5] は，統合開発環境（IDE）のコード補完が大量の候補を出力するために選択が難しいという問題に着目し，補完が実行された箇所の直前で呼び出されている API を利用して補完候補を絞り込む手法を提案した．同様に直前の API 呼び出しを利用することで，山本ら [14] は，具体的かつ大きなコード片を推薦する手法を提案した．これらの 2 つの手法では，どちらもいくつかの API 呼び出しが直前に書かれていることを前提としているため，開発者がどのようなライブラリや API を使用するかを決められない状況では，適用することができない．

そこで本研究では，Java プログラムで新規にメソッドを作成しようとしている開発者に対して，メソッド本体が書かれておらずメソッド名までのみが記述されている状態で，メソッド本体で使用する可能性の高い API 集合を推薦する手法を提案する．推薦には API 呼び出しに関連する相関ルール [4] を利用する．具体的には，メソッド名とメソッド本体の間に強い関連がある [6], [9] ことに着目し，メソッド本体での API の呼び出しと，メソッド名やクラス名などの識別子との間の相関ルールを，既存のソースコード集合から学習する．この相関ルールは，前提部がメソッド名やメソッド周辺の識別子（クラス名，フィールド名など）からなり，帰結部は本体で呼び出される API 集合である．推薦が行われる状況を満たす相関ルール集合によって，本体で呼び出される可能性が高い API 集合を得る．開発者は推薦された API 集合を見ることで，その API を呼び出して使用したり，ライブラリや API の調査の手がかりとして利用したりすることができる．

以下，2 章で提案手法を詳説し，3 章で提案手法の実装について述べる．4 章では提案手法の有効性を評価する実験とその結果を示す．5 章で関連研究を紹介し，最後に 6 章でまとめと今後の課題について説明する．

2. 提案手法

本章では，開発者が Java のメソッド名までを記述した段階で，メソッド本体で使用する可能性が高いと考えられる API の集合（メソッドの雛形）を推薦する手法を提案する．提案手法は，Java プログラムの開発者が膨大な API 集合から適切な API を選択する作業を支援することを目的としている．

図 1 に示すように，提案手法は 2 つのステップからなる．ステップ 1 はメソッド本体と識別子の関連の学習であり，推薦の実施に先立って 1 度だけ実施される．学習によって得られた知識は，相関ルールの形でデータベースに記録される．ステップ 2 はメソッド本体の雛形の推薦であり，開発者が推薦を要求するたびに実行される．雛形の推薦では，推薦が要求されたソースコードの状態を用いて，事前に構築したルール DB を参照し，雛形を生成する．雛形は複数個生成されるため，開発者には雛形のリストが提示される．

2 つのステップの詳細を，2.1 節と 2.2 節で述べる．

2.1 ステップ 1：メソッド本体と識別子の関連の学習

ステップ 1 では，既存のソースコード集合を分析することで，メソッド内での API 利用とメソッドに関連する識別子との間の関連を，相関ルールの形で抽出する．本ステップは 2 つのサブステップからなる（図 1）．サブステップ 1-a では，相関ルールマイニングに必要となるトランザクション集合を構築する．サブステップ 1-b では，構築したトランザクション集合に対して相関ルールマイニングを行い，その結果から推薦に使用できる形式の相関ルールのみを取り出して，ルール DB に保存する．

2.1.1 サブステップ 1-a：トランザクション集合の構築

このサブステップでは，相関ルールマイニングを行う準備として，相関ルールマイニングの入力となるトランザクション集合を構築する．ソースコード集合内の 1 つのメ

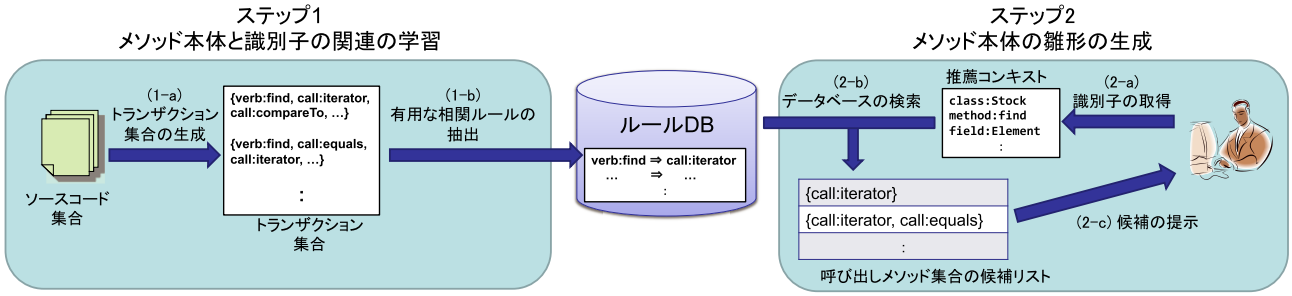

図 1 提案手法の全体像
Fig. 1 Overview of proposed method.

表 1 トランザクションの要素となる項目
Table 1 Elements in a transaction.

項目名	説明
クラス名	メソッドを所有するクラスの名前
親クラス名	メソッドを所有するクラスの親クラスの名前
インタフェース名	メソッドを所有するクラスが実装するインタフェースの名前
フィールド名	メソッド本体がアクセスするフィールドの名前
呼び出しメソッド	メソッド本体で呼び出しているメソッド（クラスの完全限定名とメソッド名とを，ピリオドで連結した文字列）
メソッド名の動詞	メソッド名の動詞部
メソッド名の目的語	メソッド名に含まれる名詞節

ソッド定義が，1 つのトランザクションに対応する．トランザクションの要素としてソースコードから取得されるデータを，**表 1** に示す．トランザクションの各要素には項目の種類を示す接頭辞が付けられるため，同名であっても種類が異なる場合には別の要素として区別される．

トランザクションの要素を取得する方法を説明する．まず，クラス名，親クラス名，インタフェース名，フィールド名，呼び出しメソッドの 5 つは，メソッド定義を含むソースコードを構文解析および意味解析することによって得られる．ソースコードの解析には，統合開発環境 Eclipse [1] の一部である ASTParser を使用する．呼び出しメソッドの取得では，ASTParser によってメソッドの所属するクラスを取得し，そのクラスの完全限定名とメソッド名とをピリオドで連結した文字列を生成する．たとえば，java.util.ArrayList 型の変数に対して add メソッドが呼び出されている場合，完全限定名とメソッド名をピリオドで連結したうえで，呼び出しメソッドの接頭辞である call:を付与した，`call:java.util.ArrayList.add` がトランザクションの要素として抽出される．

残る要素であるメソッド名の動詞と目的語の取得は，ASTParser によって取得したメソッド名に対して，以下の手順を実施する．まず，Java の標準的な命名方法である Camel Case ルールに従って，メソッド名を単語列へと分割する．次に，単語列の先頭の単語が既知の動詞であるかを 3 つの辞書（OpenNLP [2]，WordNet [3]，独自の辞書）を検索して判定し，既知の動詞であった場合にはそれをメソッド名の動詞として抽出する．独自の辞書には，我々の経験からメソッド名において実質的な動詞として使われていると判断される 7 つの単語 to，new，init，calc，cleanup，setup，shutdown が登録されている．先頭の単語が動詞として判断された場合には，残る単語列から同じ辞書を用いて名詞と判定される単語の連続を取り出し，これを目的語として抽出する．

2.1.2 サブステップ 1-b：必要な相関ルールの抽出

このサブステップでは，前サブステップで構築したトランザクション集合に対して相関ルールマイニングを行い，その結果から API 推薦に使用できる形式の相関ルールのみを取り出して，雛形 DB に保存する．

相関ルール [4] とは，「ある要素の集合 A がトランザクションに出現する場合には，そのトランザクションに別のある要素の集合 B も出現する可能性が高い」という関係を表すルールであり，$A \rightarrow B$ と表記される．前者の集合を前提部，後者の集合を帰結部と呼ぶ．相関ルールには，支持度と確信度と呼ばれる 2 つの指標値がある．

相関ルールマイニングは，トランザクション集合に含まれる任意の要素の組合せに対して相関ルールの関係を発見するが，本手法で利用できるのは，帰結部にメソッド呼び出しを持つ相関ルールのみである．なぜならば，本手法で推薦されるのはメソッド内で呼び出されうる API であるため，帰結部にメソッド呼び出し以外の要素があっても推薦時には役立たないためである．そこで，相関ルールマイニングによって得られる相関ルール集合のうち，{ 呼び出しメソッド以外のアイテム集合 } ⇒ { 呼び出しメソッドの集合 } という形式となるもののみを，ルール DB に保存する．

2.1.2.1 雛形優先度のパラメータチューニング

ルール DB を構築した直後に，ステップ 2 の実行に必要となるパラメータのチューニングを行う．チューニングの説明にはステップ 2 の詳細についての知識が必要となるため，手順の詳細は，2.2.3.1 で述べる．

2.2 ステップ 2：メソッド本体の雛形の推薦

本ステップは，メソッド名のみが記述された状態のソースコードがあるときに，開発者に対して，メソッド本体で呼びだされる可能性が高い API の集合（雛形）のリストを提示し，メソッド本体の開発を支援する．本ステップは，3 つのサブステップ (2-a) 推薦コンテキストの取得，(2-b) データベースの検索，(2-c) 候補の提示，からなる．まず (2-a) では，推薦が行われるメソッドの周辺から識別子を収集し，推薦コンテキストとして次のサブステップに渡す．(2-b) では推薦コンテキストを用いてルール DB を検索し，推薦する状況に合致する相関ルールを発見する．(2-c) では，各相関ルールの帰結部にあるメソッド呼び出しの集合を雛形とし，適切な並べ替えを行ったうえで開発者に提示する．

以下，各サブステップの詳細を説明する．

2.2.1 サブステップ 2-a：推薦コンテキストの取得

このサブステップでは，推薦が行われたメソッドの周辺から情報を収集し，次のサブステップ (2-b) へと引き渡す．

取得するデータは**表 2** に示す 6 項目である．これは表 1 に示した項目と似ているが，メソッド本体が存在しないことに起因して 2 つの違いがある．1 点目は呼び出しメソッドが含まれていないことであり，2 点目はフィールド名がクラスの持つフィールドとなっていることである．項目の違いを除けば，要素の抽出方法は 2.1.1 項で説明した手順と同じであり，項目の種類ごとに付与する接頭辞も同じ文字列を用いる．

2.2.2 サブステップ 2-b：データベースの検索

このサブステップでは，前サブステップから得た推薦コンテキストをクエリとして，ルール DB を検索して推薦の状況に合致する相関ルールを発見する．推薦する状況に合致する相関ルールとは，ルール DB に記録された相関ルールのうち，前提部がクエリの部分集合となっているルールのことである．推薦が行われる状況は，検索されたルールの前提部を満たしているため，帰結部が成り立つと推測される．

2.2.3 サブステップ 2-c：候補の提示

このサブステップでは，前サブステップで検索されたそれぞれの相関ルールから帰結部を取り出してメソッド本体の雛形を作る．一般に，検索では複数のルールが発見されるため，雛形もまた複数のものが生成される．そのため，雛形の集合は適切な順序で並べ替え，雛形のリストとして開発者に提示される．

雛形は，呼び出しメソッドの集合そのままであるため，特別な処理は行わない．

雛形の並べ替えでは，開発者が使用する可能性が高い雛形を上位に置く必要がある．そこでいくつかの基準を組み合わせた優先度を設定し，その値の大きい順にリストを並べ替えることとする．優先度の計算には，相関ルールの性質である以下の 4 つの変数を用いる．

B_r：支持度
C_r：確信度
N_r：前提部の要素数
M_r：帰結部の各呼び出しメソッドの IDF [8] の総和（学習に用いたメソッド全体を文書集合とする）

各指標を採用した理由を説明する．B_r 支持度と C_r 確信度は，相関ルールの信頼性を表す指標であり，どちらの指標も高い値であるほど帰結部が成り立つ可能性が高いと考えられるためである．前提部の要素数が多いほど，多くの要素が推薦の状況にあてはまっていることを示しているためである．最後の，帰結部の各呼び出しメソッドの IDF の総和は，推薦されるメソッド（API）の珍しさを表しており，この値が大きいほど推薦を受けたときに開発者が得る情報量 [11] が多いと考えられるためである．

優先度 P_r は下の式で定義され，4 つの変数を重みつきで足し合わせることによって求める．

$$P_r = bB_r + cC_r + nN_r + mM_r$$

2.1.2.1 で説明したように，4 つのパラメータ b，c，n，m はルール DB を構築した直後に行われるパラメータチューニングによって決定する．

2.2.3.1 雛形優先度のパラメータチューニングの手順

4 つのパラメータ b，c，n，m を決定するために，ルール DB の構築直後にパラメータチューニングを行う．まず，学習に使用したものとは重複しないソースコード集合（チューニング集合）を用意し，チューニング集合に含まれるすべてのメソッド定義に対して，メソッド本体を削除したうえで雛形の集合を算出する．削除されたメソッド本体で実際に呼び出されていたメソッドの集合を正解とし，雛形リストが正解に近づくようなパラメータを発見する．

雛形リストと正解と近さは，下で説明する指標で評価する．ソースコード集合中のメソッド定義の集合 M と，メソッド定義 $m \in M$ が与えられたときに候補リスト $L(m)$ を生成するリスト生成器 L が与えられたとき，評価値 $E(M, L)$ は以下のように計算できる．ここで $C(m)$ は，あるメソッド定義 $m \in M$ のメソッド本体に記述されている

表 2 推薦時にソースコードから取得される項目
Table 2 Elements extracted from source code when suggesting.

項目名	説明
クラス名	メソッドを所有するクラスの名前
親クラス名	メソッドを所有するクラスの親クラスの名前
インタフェース名	メソッドを所有するクラスが実装するインタフェースの名前
フィールド名	メソッド本体を所有するクラスに所属するフィールドの名前
メソッド名の動詞	メソッド名の動詞部
メソッド名の目的語	メソッド名に含まれる名詞節

呼び出しメソッド集合であり，重複したメソッド呼び出しは排除されている．

$$E(M,L) = \sum_{m \in M} \sum_{c \in C(m)} \frac{\mathrm{wratio}(c,m)}{\mathrm{maxRank}(c, L(m))}$$

式中の $\mathrm{wratio}(c,m)$ 関数は，メソッド m における，呼び出しメソッド c の重み付けされた割合であり，IDF を用いて以下のように定義する．

$$\mathrm{wratio}(c,m) = \frac{\mathrm{IDF}(c)}{\sum_{d \in C(m)} \mathrm{IDF}(d)}$$

定義より，任意のメソッド m について，$\sum_{d \in C(m)} \mathrm{wratio}(d,m) = 1$ という関係が成立する．wratio 関数の設計意図は，メソッド m で特異的に用いられているメソッドを正しく推薦したときに評価値を大きく増加させる一方で，equals や toString のように多くのメソッドで呼び出される汎用的で著名なメソッドが正しく推薦されたときの評価値の増加を小さくとどめることである．

式中の $\mathrm{maxRank}(c, L(m))$ 関数は，候補リスト $L(m)$ の上から何番目に c が出現するかを取得する関数である．$L(m)$ はメソッド本体の雛形のリストであり，メソッド本体の雛形は呼び出しメソッドの集合でできている．

実際に使用するパラメータは，それぞれのパラメータを $[0,1]$ の範囲で細かく分割し，すべての組合せで $E(M,L)$ が最も大きくなったパラメータの組合せを使用する．

3. 提案手法の実装

提案手法を，Eclipse IDE の Java 開発環境で動作するツールとして実装した．ツールが有効となった Eclipse において，Java のメソッドの名前のみが記述された状態でメソッド名の直後にカーソルを合わせて Eclipse の持つコード補完機能（コードアシスト）を起動すると，提案手法によってメソッド本体の雛形リストが，コード補完の候補として提示される．リストから雛形を 1 つ選択することで，コメントに API が記述されたメソッド本体が生成される．

ツールの動作を，具体例とスクリーンショットを用いて説明する．開発者が TestDatabase というクラスを編集している場面を考える（図 2）．TestDatabase クラスにはいくつかのフィールドが宣言されており，ここに開発者は新しいメソッド test を作成しようとしている．このとき，開発者がコードアシストを起動すると提案手法が実行され，作成しようとしているメソッドの本体で使用される可能性の高い API の集合が，リストとして提示される（図 3）．このリストから 1 つを選択すると，選択された候補の API がコメントとして記述されたメソッド本体が，雛形として生成される（図 4）．1 行のコメントが 1 つの API 呼び出しに対応しており，=> という文字列の右側がメソッド名を，左側がメソッドの所属するクラスの完全修飾名を表す．

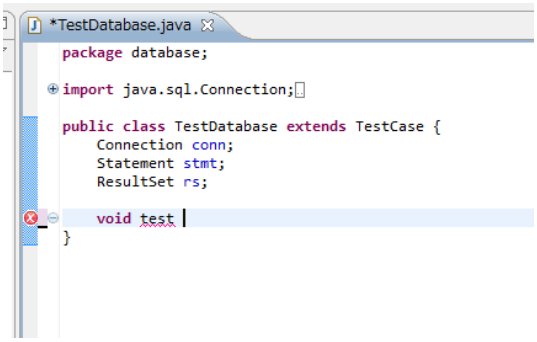

図 2 ツールを利用できる場面（メソッド名と戻り値が書かれた状態）
Fig. 2 Usage scenario: (1) Invoke proposed method when method name and return type is written.

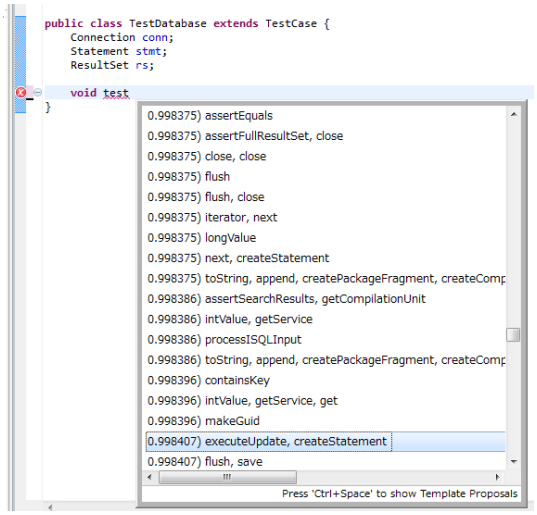

図 3 ツールを起動すると，API の集合がリストとして提示される
Fig. 3 Usage scenario: (2) List of API sets are suggested.

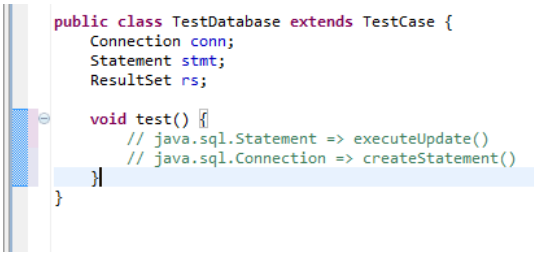

図 4 API 集合を 1 つ選択すると，メソッドの雛形が入力される
Fig. 4 Usage scenario: (3) Select one element from the list, then a template of method body is inserted.

4. 実験

本手法によって推薦される API 集合が，開発者にとって有用なものであるかどうかを調査するために，評価実験を行った．本手法では，記述中のソースコードにおいて必要な API が分からない開発者に対して，そのソースコードで記述されるであろう API 集合を提案することで，開発者に手がかりを与えることを期待している．そこで本実験では，開発者に有用な API 集合を推薦できるか調査するた

めに，以下の 3 つのリサーチクエスチョン（RQ）を設定した．

RQ1：推薦された API 集合はメソッド本体で使用されそうな API 集合であるか．

RQ2：ソースコードから取得したコンテキストのうち，どの項目が有用な推薦の手がかりとして使用されたか．

RQ3：直前に書かれた呼び出しメソッドの列から次に書かれそうなコードの候補を推薦する既存手法との連携が期待できるか．

4.1 方法と結果

本手法を実装したツールで，開発者にメソッド本体で使うべき API の手がかりを与えられるか調査するために，学習に使用したソースコードとは別のソースコードに対して本ツールで API 集合の推薦を行い，実際にメソッド本体に記述されている呼び出しメソッドとの比較や，候補に使用した相関ルールを調査する方法を用いた．

学習に用いたソースコード集合は，ソフトウェア検索システム SPARS TYPE:R [7] の検索対象であり，プロジェクト数は 443，ファイル数は 198,324 である．ここから，学習によって 5,401,676 の相関ルールを得た．並べ換え優先度のパラメータチューニングは，学習に用いた Java ファイルとは異なる 10,000 の Java ソースファイルを用いて実施し，パラメータを $b = 0.0007425$，$c = 0.001502$，$n = 1.000$，$m = 0.1209$ と決定した．

評価には，学習およびパラメータチューニングとは異なるソースコード集合を用いる．まず，オープンソースソフトウェアのプロジェクトホスティングサイトから，517 プロジェクトの，310,166 ファイルを収集し，ここからランダムに 10,000 ファイルを取り出す．含まれていたメソッド数は 85,390 個であり，その中からボディで 1 つ以上のメソッド呼び出しを行っていた 52,651 個のメソッドを評価の対象とする．評価対象の 1 メソッドあたりの呼び出しメソッド数は，平均 7.0 個であり，中央値は 3 であった．

4.1.1 RQ1：推薦された API 集合はメソッド本体で使用されそうな API 集合であるか

評価用のソースコードに対して本ツールで API 集合の推薦を行い，実際にメソッド本体に記述されていた呼び出しメソッドをどれくらい再現できるかを評価した．手順としては，まず，評価用のソースコードのメソッド定義からメソッド本体を削除し，そのメソッドに対して本ツールで推薦を行う．そして，推薦された API 集合のリストと，削除したメソッド本体で呼び出されていた API とを比較することで評価を行う．

まず，ツールが出力した候補リストの概要を**表 3** に示す．半数以上の事例で，候補リストに正しい呼び出しメソッドを含んでいた一方で，2,979 個（全体の 5.7%）のメソッドに対しては 1 つも候補を提示することができなかった．1 つ以上の候補を提示した 49,672 のメソッドにおいては，候補リストの長さの平均は 1,440，中央値は 526 であった．

表 3 推薦された候補リストの概要
Table 3 Summary of suggested API lists.

内訳	候補リストの分類
2,979	1 つも候補が含まれなかった
21,837	1 つ以上の候補が候補リストに含まれたが，その中に 1 つも正しい呼び出しメソッドが含まれなかった
27,835	1 つ以上の正しい呼び出しメソッドが含まれていた

表 4 候補リストの一部と全体における適合率・再現率
Table 4 Precision and recall of suggested API lists.

	適合率 1	適合率 2	適合率 3	再現率
1 位	0.128	0.133	0.133	0.031
1〜5 位	0.109	0.102	0.074	0.078
1〜10 位	0.096	0.089	0.058	0.103
1〜30 位	0.082	0.068	0.036	0.148
全体	0.053	0.027	0.008	0.290

次に，情報検索システムの評価で広く用いられている適合率と再現率を用いて，候補リストを評価する．ここでは，候補の並べ替えを評価することを目的として，一定の順位までの候補に限定した適合率および再現率を求める．ここで問題となるのは，本手法の出力（推薦結果）が集合のリストであり，一般的な情報検索の出力とは異なるために，一般的な適合率と再現率の定義を適用できないことである．そこで，適合率と再現率の定義を修正して用いることとする．また，適合率については開発者が候補リストを利用する際の観点を考慮して，3 種類の基準を用いることとした．以下に，具体的な適合率と再現率の定義を示す．

適合率 1：提示した API 集合候補のうち，正解となる API を 1 つ以上含んでいた候補の割合

適合率 2：提示した API 集合候補に含まれる API をすべて集めた多重集合を構成し，その中で正解となる API が占める割合

適合率 3：提示した API 集合候補に含まれる API を重複を除いて集めた集合を構成し，その中で正解となる API が占める割合

再現率：全正解 API 中，API 集合候補に含まれる API の割合

候補リストのトップからそれぞれ 1 位，5 位，10 位，30 位までについて，各適合率と再現率を求めた結果を，**表 4** に示す．また，適合率 1，2，3 と再現率から計算した F 値を**表 5** に示す

4.1.2 RQ2：ソースコードから取得したコンテキストのうち，どの項目が有用な推薦の手がかりとして使用されたか

推薦コンテキストのうち，どの項目が良い推薦結果を出力するのに貢献したかを知るために，候補リスト中でメ

表 5 候補リストの一部と全体における F 値
Table 5 F-measure values of suggested API lists.

	適合率 1 に基づく F 値	適合率 2 に基づく F 値	適合率 3 に基づく F 値
1 位	0.050	0.050	0.050
1〜5 位	0.091	0.088	0.076
1〜10 位	0.100	0.096	0.075
1〜30 位	0.106	0.093	0.058
全体	0.090	0.049	0.011

表 6 正解を含む相関ルールにおける，前提部の識別子の種類の割合
Table 6 Elements of precondition in association rules containing at least one correct API.

項目	割合
メソッド名の動詞	84.3%
メソッド名の目的語	2.0%
クラス名	11.0%
親クラス名	11.0%
インタフェース名	8.6%
フィールド名	5.4%

ソッド本体と一致する候補を出力することに貢献した相関ルールを取り出し，前提部にどの項目を含むかを調査した．前提部の項目の種類の割合を**表 6** に示す．各前提部は複数の項目からなるため，割合の合計は 100%とはならない．

4.1.3 RQ3：直前に書かれた呼び出しメソッドの列から次に書かれそうなコードの候補を推薦する既存手法との連携が期待できるか

もし提案手法がメソッドの前半で呼び出す API を推薦できたとすると，それに続くメソッド呼び出しを既存手法 [5], [14] によって得られると期待される．そこで，メソッド本体の前半に記述されている呼び出しメソッドが，本手法でどれくらい提案できているかを調査した．具体的には，メソッド本体に 2 つ以上の呼び出しメソッドが記述されているメソッド定義を対象として，正解メソッドの前半分（小数点以下切り捨て）すべてが含まれるメソッド定義はどの程度あるかを調査した．

まず，評価対象ソースファイル 10,000 には，2 つ以上の API 呼び出しを含むメソッドが 85,389 個存在した．提案手法により，前半部分の呼び出しメソッドすべてを候補リストによって推薦できたのは 6,945 個であり，これは全体の約 8.1%であった．

4.2 評価と考察

表 4 の結果を見ると，どの適合率の定義においても，候補の順位が上がるにつれて適合率の値が減少する傾向にある．これは，上位の候補に正解となる呼び出しメソッドが多く含まれていることを示しており，このことから本手法で提案した並べ替え基準は一定の妥当性を持っていると考えられる．しかし，本手法と直接比較できる既存研究が存在しないため，表 4 や表 5 の値について，良いか悪いかを評価することは難しい．ここで，提案手法は API の利用実績に基づいて推薦を行うため，学習に用いたソースコード集合で利用されなかった API を推薦することは，原理的に不可能である．この事実をふまえると，29%という再現率は良いものであると考えている．以上から，RQ1 については，推薦された API 集合はメソッド本体で使用されそうな API 集合を含んでいたと結論づけたい．

推薦コンテキストの項目が良い推薦につながったかを評価した表 6 を見ると，項目によって値の違いが大きいことが分かる．なかでも，メソッド名の動詞は 84.3%と特に大きい値となっており，メソッド名の動詞が果たした役割が大きいことが分かる．この結果は，メソッド名とメソッド本体の関係を調べた先行研究 [6], [9] の知見とも一致する．一方で，動詞以外の項目が果たした役割は小さいため，ここを工夫することで推薦リストを改善する余地があるのではないかと考えられる．たとえば，本手法ではクラス名やインタフェース名は名前の内部構造には触れず，識別子名が一致することをルールマッチの条件とした．しかし，クラス名もまた複数の単語の組合せからなることが多いため，その名前が完全に一致することは稀である．クラス名などの識別子についても，それを構成する単語に分割することで，API 利用との関連を見つけることができるかもしれない．

RQ3 については，前半の呼び出しメソッドを推薦できた候補が全体の約 8.1%と低い割合であったため，現時点では既存手法と直接的に連携することは難しいと考えられる．学習と推薦をメソッド本体の前半で行われる API 呼び出しに絞って行うなどの最適化によって，連携が可能になるかもしれない．

評価対象のメソッドを調査したところ，評価対象のソースコード集合で記述された呼び出しは全部で 344,250 個であったが，そのうち 61,754 個（約 18%）は同一クラス内で定義されたメソッドであり，さらにそのうち 17,007 個（全体の約 5%）はプライベートメソッドであった．提案手法ではこのような呼び出しを推薦することは原理的に不可能であるため，再現率や適合率の評価値を下げる原因となっている．さらに，このような API は開発者が変更しているメソッドに近い場所で定義されているものであるため，提案手法によって推薦する必要性も低いと考えられる．今後は，クラス内などの近い場所で定義されたメソッド呼び出しを評価対象から除くとともに，近い場所で定義されたメソッドの利用を支援する方法についても考える必要がある．

5. 関連研究

山本らは，大規模なソースコード集合から適切なコード片を推薦する手法を提案した [14]．この手法は，IDE で書きかけのソースコード片を入力として，そのソースコード片に対応した残りのソースコード片を頻度の多い順に出力

するという手法で推薦を行う．山本らの研究では，メソッド本体に記述された呼び出しメソッドの列から次に書かれそうなコード片を推薦するのに対し，本研究では，メソッド本体に何も記述されていない状態で推薦を行う．また，山本らの研究で挿入されるコードはそのままメソッド本体で使用できる完全なコード片なのに対し，本研究で挿入されるコードは編集が必要な雛形である点も異なる．

Wang らは，開発者があるメソッドを呼び出すよう記述した際に，そのメソッドの利用例と説明文を提示する手法APIExample を提案した [12]．この手法はまず，開発者が記述したメソッドの完全限定名に，java と example の 2 語を加え，Google を用いてウェブページ検索を行う．そして，検索結果の上位 300 件のページからコード片と周囲のテキストを抽出し，クラスタリングしたうえで開発者に提示する．Wang らの手法は特定のメソッドの利用例を示すのに対し，提案手法はある名前を持つメソッドの中で呼び出すメソッドの候補を提示する点が異なる．また，Wang らの手法が利用する情報源が一般のウェブページであるのに対し，提案手法の情報源は既存のソフトウェアのソースコードである点も異なる．

Monperrus らは，オブジェクト指向言語のソースコード中で不足している呼び出しメソッドの自動検出手法を実装した DMMC [10] というツールを提案している．この手法では，ある程度記述されたソースコードに対して必要であろう API を提案するが，本手法では，これから記述する新規のメソッドに対して API を提案する点が異なる．

Ye ら [13] や島田ら [15] は IDE 内で使用できるコード片推薦手法を提案している．これらの手法では，開発者がソースコードを記述している IDE 内から自動的に抽出したキーワードに関連するソースコードを検索する．本手法では，開発者が新規作成するメソッドの本体に対して API 集合を推薦する．

6. まとめと今後の課題

本研究では，膨大な API の中から必要な API を選択し，その組合せを考えることを支援する手法として，開発者が新規に作成するメソッドの名前を記述した段階で，メソッド本体で使用される可能性が高い API 集合を推薦する手法を提案した．本手法によって開発者に有用な API 集合を推薦できるかを調査したところ，推薦の際に並べ替えによって適切な候補が上位に並べられていることや，開発者の記述中のソースコードに，より適した API 集合を推薦する改善案などが分かった．

今後の課題としては，API を集合としてではなく，呼び出し順序を含めた列として開発者に提示することで，API の組合せについても支援することを検討している．また，メソッド本体にある程度コードを書いた状態で使用するコード補完手法と連携することで，より精度の高いコードの推薦を行うことも考えている．

謝辞 本研究は JSPS 科研費 25730036，25220003，26280021 の助成を受けたものです．

参考文献

[1] Eclipse, available from ⟨http://www.eclipse.org/⟩.
[2] OpenNLP, available from ⟨http://opennlp.apache.org/⟩.
[3] WordNet, available from ⟨http://wordnet.princeton.edu/⟩.
[4] Agrawal, R., Imieliński, T. and Swami, A.: Mining Association Rules Between Sets of Items in Large Databases, *Proc. 1993 ACM SIGMOD International Conference on Management of Data*, pp.207–216 (1993).
[5] Bruch, M., Monperrus, M. and Mezini, M.: Learning from Examples to Improve Code Completion Systems, *Proc. 7th Joint Meeting of the European Software Engineering Conference and the ACM Symposium on the Foundations of Software Engineering* (*ESEC/FSE 2009*), pp.213–222 (2009).
[6] Høst, E.W. and Østvold, B.M.: Debugging Method Names, *Proc. 23rd European Conference on Object-Oriented Programming* (*ECOOP 2009*), pp.294–317 (2009).
[7] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Trans. Softw. Eng.*, Vol.31, No.3, pp.213–225 (2005).
[8] Jones, K.S.: A statistical interpretation of term specificity and its application in retrieval, *Journal of Documentation*, Vol.28, pp.11–21 (1972).
[9] Kashiwabara, Y., Onizuka, Y., Ishio, T., Hayase, Y., Yamamoto, T. and Inoue, K.: Recommending Verbs for Rename Method using Association Rule Mining, *Proc. IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering* (*CSMR-WCRE*) (2014).
[10] Monperrus, M., Bruch, M. and Mezini, M.: Detecting Missing Method Calls in Object-Oriented Software, *Proc. 24th European Conference on Object-Oriented Programming*, pp.2–25 (2010).
[11] Shannon, C.E.: A Mathematical Theory of Communication, *The Bell System Technical Journal*, Vol.27, pp.379–423, 623–656 (online), available from ⟨http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf⟩ (1948).
[12] Wang, L., Fang, L., Wang, L., Li, G., Xie, B. and Yang, F.: APIExample: An effective web search based usage example recommendation system for java APIs, *Proc. 26th IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2011*), pp.592–595 (2011).
[13] Ye, Y., Fischer, G. and Reeves, B.: Integrating Active Information Delivery and Reuse Repository Systems, *Proc. 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-first Century Applications*, pp.60–68 (2000).
[14] 山本哲男，吉田則裕，肥後芳樹：ソースコードコーパスを利用したシームレスなソースコード再利用手法，情報処理学会論文誌，Vol.53, No.2, pp.644–652 (2012).
[15] 島田隆次，市井 誠，早瀬康裕，松下 誠，井上克郎：開発中のソースコードに基づくソフトウェア部品の自動推薦システム A-SCORE，情報処理学会論文誌，Vol.50, No.12, pp.3095–3107 (2009).

早瀬 康裕 （正会員）

平成 14 年大阪大学基礎工学部情報科学科卒業．平成 19 年同大学大学院博士後期課程修了．同年同大学特任助教．平成 22 年東洋大学総合情報学部助教．平成 23 年筑波大学システム情報系助教．博士（情報科学）．ソフトウェア開発支援，オープンソースソフトウェア開発分析の研究に従事．IEEE 会員．

鬼塚 勇弥

平成 24 年大阪大学基礎工学部情報科学科卒業．平成 26 年同大学大学院情報科学研究科博士前期課程修了．現在，株式会社カプコンに勤務．在学中はコード補完手法に関する研究に従事．

山本 哲男 （正会員）

平成 9 年大阪大学基礎工学部情報工学科卒業．平成 14 年同大学大学院博士後期課程修了．同年科学技術振興事業団計算科学技術研究員．平成 16 年立命館大学情報理工学部情報システム学科講師．平成 20 年同学科准教授．平成 23 年日本大学工学部情報工学科准教授．博士（工学）．ソフトウェア開発支援環境の研究に従事．電子情報通信学会，IEEE-CS 各会員．

石尾 隆 （正会員）

平成 15 年大阪大学大学院基礎工学研究科博士前期課程修了．平成 18 年同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）．同年ブリティッシュコロンビア大学ポストドクトラルフェロー．平成 19 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻助教．博士（情報科学）．プログラムの静的解析と動的解析，プログラム理解に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

井上 克郎 （フェロー）

昭和 31 年生．昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院博士課程修了．同年同大学基礎工学部助手．昭和 59～61 年ハワイ大マノア校情報工学科助教授．平成元年大阪大学基礎工学部講師．平成 3 年同助教授．平成 7 年同教授．平成 14 年大阪大学大学院情報科学研究科教授．平成 24 年大阪大学大学院情報科学研究科・研究科長．工学博士．ソフトウェア工学，特に，ソフトウェア開発手法，プログラム解析，再利用技術の研究に従事．