メソッド周辺の識別子名とメソッド本体のAPI利用実績に基づいたAPI集合推薦手法の提案と評価

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

概要：開発者はソフトウェア開発では，必要な API を選択し，それをどのように組み合わせる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であろう API を推薦することで，API の選択を支援する手法を提案する．開発者は推薦された API を参考にしてメソッド本体を編集する．API の推薦にはメソッド名やクラス名，フィールド名といったメソッド周辺の識別子を使用する．推薦する API の学習は大規模なソースコード集合の情報に相関ルールマイニングを用いて行う．本手法によって開発者に有用な API 集合を推薦できるか調査したところ，推薦の際に並び替えによって適切な候補を上位に並べられていることや，より開発者の記述中のソースコードに適した API 集合を推薦する改善案などを確認した．

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 APIs for a Empty Method Based on Surrounding Identifiers and API Usage History

Abstract: In modern software development, developers have to select and combine appropriate APIs from software libraries to implement any 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

近年，ソフトウェア開発が大規模化，複雑化する一方で，品質の高いソフトウェアを効率的に開発することが求められている．その理由として，医療や金融といった社会の重要な場面でソフトウェアを利用するようになったことや，急速な技術の進歩に遅れを取らないため納期が短くなっていることなどが挙げられる．

ソフトウェア開発では，多種多用な Application Programming Interface (API) を組み合わせる必要がある．これは，ソフトウェアが複数のライブラリやフレームワークを組み合わせて開発され，それらの機能は API を通じて利用するためである．例えばクライアントから入力されたSQL クエリによってデータベースを管理する Web アプリケーションであれば，サーブレットに送られてきたリクエストの情報を取得するためのライブラリ，受け取ったリクエストの SQL クエリを正しく実行できるように文字列処理を行うためのライブラリ，SQL を実行してデータベースの更新を行うためのライブラリといった複数のライブラリが利用され，開発者はそれぞれのライブラリの API を組み合わせて開発を行う．

しかし，多種多様な API の中から利用すべき API を選

[1] 大阪大学大学院情報科学研究科
[2] 筑波大学システム情報系
[3] 日本大学工学部情報工学科
[a)] y-onizuk@ist.osaka-u.ac.jp
[b)] hayase@cs.tsukuba.ac.jp
[c)] tetsuo@cs.ce.nihon-u.ac.jp
[d)] ishio@ist.osaka-u.ac.jp
[e)] inoue@ist.osaka-u.ac.jp

択し，それらを適切に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．API を用いて開発を行う場合，現在では非常に多くのライブラリやフレームワークが提供されているため，まずはそこから必要な機能を持つものを選択する必要がある．そして，一つのライブラリやフレームワークでも複数の機能を持つ場合が多く，必要な機能がどの API によって提供されているかを探さなければならない．更に，使用すべき API が定まってもそれをどのように使うかを知るために，ドキュメントやサンプルコードを調査する必要がある．このように，利用すべき API がわからない状態から実際にその API を使用するまでには複数の作業が必要になり，それぞれに難しさや手間がある．

このような問題を解決するアイデアとして，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手法が提案されている [4], [11] ．これらの手法は，候補の提示に開発者が記述中のソースコードの文脈を利用するため，標準的な統合開発環境 (IDE) が持つコード補完機能よりも適切な候補を推薦することができる．しかし，これらの手法では，数個の API を使用することを開発者が決め，呼び出しメソッド文として記述した後でなければ，適切な候補の推薦を行うことができないため，メソッド本体で使用する API がわからない場合や思い付かない場合に利用するのが難しい．

そこで本研究では，メソッドを新規作成しようとしている開発者に対して，メソッド名を記述した時点で，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．開発者がメソッド本体で使用する可能性の高い API 集合は，メソッド名とメソッド本体に強い関連があること [5], [7] に着目し，既存のソースコードに対して相関ルールマイニングを行い学習した API 利用実績に基づいて推薦する．本研究では，メソッド名とメソッド本体だけでなく，メソッド周辺の識別子としてクラス名，フィールド名の情報も API 集合を推薦する手掛かりとして利用する．API 集合はメソッド本体の雛形として提供され，開発者はその雛形を調査の手掛かりとして利用したり，雛形に編集を加えることでメソッド本体を完成させる．

2. 背景

本章では，メソッド周辺の識別子名から API 利用実績を学習できる根拠として，メソッド名とメソッド本体に強い関連があることを述べる．その後，API 利用実績の学習に利用する相関ルールマイニング技術について説明する．

2.1 メソッド名とメソッド本体の関係

メソッド名に使われる動詞は，メソッド本体で利用している API や使用しているデータの種類，メソッド外部の情報に強く関係していることが，Host ら [5] や柏原ら [7] の研究によって明らかになっている．例えば，メソッド名の動詞に create が使用されているメソッドの本体では，API に Instance という単語が使用されていることが多く，メソッド名の動詞に find が使用されているメソッドの本体では，API として hasNext と iterator が同時に使用されていることが多い．

2.2 相関ルールマイニング

相関ルールマイニングとは，蓄積された大量のトランザクションから，同時性や関係性が強い事象間の関係 (相関ルール) を抽出する技術である [3] ．相関ルールには，ある事象の下である事象が発生するという関係と，トランザクション集合中にその関係が出現する割合などが含まれるため，相関ルールマイニングによって，大量のデータから，同時に出現する要素やその出現確率などを調べることができる．

3. 提案手法

本研究では，膨大な API から開発するソフトウェアで使用する機能に必要な API を選び出すのが難しいという問題を解決するために，メソッド名を記述した開発者に対して，開発者がメソッド本体で使用するであろう API の集合を推薦する手法を提案する．API の集合を推薦することで，開発者がメソッド本体で使うべき API の手掛かりを与え，API の選択に必要な調査の労力を軽減する．

本手法は，作成したいメソッドの本体でどのような API を呼び出せばよいかわからない開発者が使用することで，メソッド本体の記述や必要な情報の検索の手掛かりを提供する．そこで，開発者には API 集合として呼び出しメソッドを推薦する．呼び出しメソッドを推薦するのは，メソッド本体が呼び出しメソッドの組み合わせで構成されていること，そして用いる呼び出しメソッドがわかればその使い方をインターネット検索や API 利用方法推薦システム [6], [9] で調べられることに基づく．本論文では，推薦する呼び出しメソッドの集合を，メソッド本体の雛形と呼ぶ．

記述したメソッド名に適したメソッド本体の雛形の推薦には，既存のソースコードの API 利用実績を使用する．そこで，メソッド名とメソッド本体に強い関連があること [5], [7] に着目し，既存のソースコードの識別子に対して相関ルールマイニングを行い学習した API の利用実績に基づいて，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．相関ルールマイニングでは，メソッド名とメソッド本体だけでなく，メソッド周辺の識別子としてクラス名，フィールド名の情報も API 集合を推薦する手掛かりとして利用することで，より記述中のソースコードに適した API 集合を推薦を目指す．

提案手法は，図 1 の 2 ステップからなる．メソッド本体と識別子の関連の学習は，メソッド本体の雛形の推薦の前に事前に行っておく必要がある．

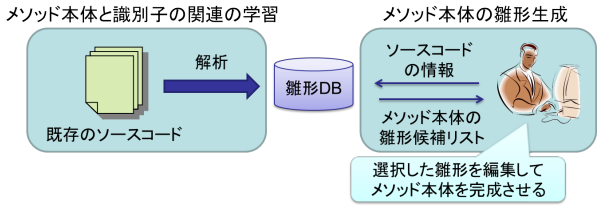

図 1　提案手法の概要

3.1　a) メソッド本体と識別子の関連の学習

メソッド本体と識別子の関連の学習では，既存のソースコードから，本手法の推薦に利用するための API 利用実績を抽出する．メソッド本体と識別子の関連の学習は，以下の 2 ステップからなる．

(1) トランザクション集合の生成
(2) 相関ルールの抽出

トランザクション集合の生成では，学習に使用する既存のソースコードの構文解析を行い，メソッド名や呼び出しメソッドといった定義メソッドの情報，定義クラスやフィールドといったメソッドの周りの識別子から，相関ルールマイニングに使用するトランザクション集合を生成する．相関ルールの抽出では，生成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，その出力から推薦に使用する相関ルールのみを抽出する．

3.1.1　Step a-1. トランザクション集合の生成

API 集合を推薦する手掛かりとなる識別子に対して相関ルールマイニングを行う準備をする．そこで，大規模なソースコード集合に構文解析を行って識別子を取得し，相関ルールマイニングに使用するトランザクションを生成する．入力は既存のソースコード集合であり，出力はトランザクション集合である．

相関ルールマイニングに使用するトランザクションは，ソースコード中のメソッド定義 1 つごとに 1 つ生成できる．解析対象のメソッド集合を M，あるメソッド m のコンテキストを X(m) とすると，相関ルールマイニング用のトランザクション T は次のようなコンテキストの集合として記述できる．

$$T = X(m) \mid m \in M$$

メソッドのコンテキストとは，1 つのメソッド周辺に出現した識別子の集合であり，1 つのメソッドに対して一意に定まる．コンテキストに含まれる要素は，表 1 に示す 7 種類である．

本ステップでは，入力された大規模なソースコード集合に構文解析を行って表 1 の 7 種類の識別子を取得し，各メソッド定義ごとにトランザクション T を生成してその集合を出力する．構文解析には Eclipse のプラグイン開発ライブラリで提供されている ASTParser を使用した．

メソッド名は主語と動詞を持った完全な英文ではないことが大半であり，正確に品詞解析を行うことが難しいため，メソッド名の動詞と目的語の抽出は以下の手順で行う．まずメソッド名を単語ごとに分割して品詞解析を行い，動詞と名詞句を抽出する．品詞解析した結果に動詞がみつからなければ，メソッド名の先頭の単語に対して辞書を引くことで品詞の確認を行い，動詞の品詞を持つ単語であればそれを動詞とする．この操作には，品詞解析ツール OpenNLP [1] と，辞書 WordNet [2] を使用した．また，メソッド名先頭の to，new，init，calc，cleanup，setup，shutdown の 7 つの単語については， Java のメソッド名において慣習的に動詞として用いられるため，品詞解析の結果に関わらず動詞とする．最後に，得られた動詞をメソッド名の動詞，名詞句を目的語として出力する．

3.1.2　Step a-2. 相関ルールの抽出

既存のソースコードに出現する識別子間の相関ルールを抽出することで，推薦に用いる API 利用実績を取得する．API の利用実績の学習には，Step a-1 で得られたトランザクション集合に相関ルールマイニングを行う．入力は Step a-1 で得られたトランザクション集合であり，出力は { 呼び出しメソッド名以外のアイテム集合 } ⇒ { 呼び出しメソッド名の集合 } となっている相関ルールの集合である．

本ステップでは，まず Step a-1 で得られ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，その後相関ルールマイニングで得られた相関ルールのうち，帰結部が呼び出しメソッド，条件部がそれ以外であるような相関ルールを抽出する．相関ルールは (条件部，帰結部，支持度，確信度) の 4 種類の情報からなる．

3.2　b) メソッド本体の雛形の推薦

メソッド本体の雛形の推薦では，メソッド名を記述した開発者にメソッド本体の雛形を推薦し，開発者の API 選択を支援する．メソッド本体の雛形の推薦は，以下の 3 ステップからなる．

(1) 識別子の取得
(2) データベースの検索
(3) 候補の提示

これは，開発者が新規作成したいメソッド名を入力し終えたタイミングで開始されるものである．識別子の取得で

表 1　学習のためにソースコードから取得する識別子

識別子	内容
メソッド名の動詞	メソッド名に使われている動詞
メソッド名の目的語	メソッド名に使われている名詞節の集合
クラス名	メソッド定義クラスの名前
親クラス名	メソッド定義クラスの親クラス名
インタフェース名	メソッド定義クラスのインタフェース名
呼び出しメソッド名	メソッド内で呼び出しているメソッド名
フィールド名	本体で使用している全フィールドの名前

は，開発者が編集しているソースコードに構文解析を行い，関連の学習で使用した識別子を取得する．データベースの検索では，得られた識別子集合で雛形 DB の相関ルールを検索する．候補の提示では，得られた相関ルールから雛形を生成し，並べ替えを行って開発者に提示する．

3.2.1 Step b-1. 識別子の取得

記述中のソースコードから，API 利用実績の検索に用いる識別子として，メソッド本体で使用されそうな API 集合を決定する手掛かりとなる識別子を取得する．入力は開発者が記述中のソースコードであり，出力は情報源と識別子の組の集合である．取得するアイテムは表 2 に示す 6 種であり，これは表 1 のアイテムのうち記述中のソースコードから取得できるものである．Step a-1 と同様，メソッド名には品詞解析を行い動詞と目的語を取得する．

3.2.2 Step b-2. データベースの検索

メソッド本体で使用されそうな API 集合を決定する手掛かりとなる識別子で雛形 DB を検索し，記述中のソースコードと関連の強い相関ルールを取得する．入力は Step b-1 で得られたアイテム集合であり，出力は相関ルール集合である．

ここで検索する相関ルールは，雛形 DB に格納されたルール集合を R，開発者が記述中のソースコードから得られたアイテム集合を $Q = \{q_1, q_2, \ldots, q_n\}$，相関ルール $r \in R$ の条件部を A_r としたとき，以下の条件を満たす相関ルール r である．

$$\{r \mid r \in R \wedge A_r \subseteq Q\}$$

3.2.3 Step b-3. 候補の提示

記述中のソースコードに適したメソッド本体の雛形を提示し，開発者の API 選択を支援する．入力は Step b-2 で得られた相関ルール集合であり，出力はメソッド本体の雛形リストである．まず，相関ルールの帰結部の呼び出しメソッド集合をソースコードに挿入する文字列に変換し雛形を生成する．その後，メソッド本体の雛型を並び替えてリストを作成し，開発者に提示する．

メソッド本体の雛形の並び替えは，開発者が使用する可能性が高い雛形や，開発者に与える情報が大きい雛形ができるだけ上位に出現するような順にしたい．そこで，新規作成するメソッドの本体と関係が強いであろう相関ルールの基準をいくつか考え，それら基準を組み合わせて各相関ルールの優先度を決定して並び替える．並び替えのために筆者が考えた基準は，1) 支持度が高い，2) 確信度が高い，3) 条件部の要素数が多い，4) 帰結部の要素数が多い，5) 帰結部に出現するアイテムの希少度が高い の 5 つである．基準 1, 2 は，その相関ルールが学習に使用したソースコード集合でどの程度頻繁に出現するかを示す値として，基準 3 は記述中のソースコードと相関ルールの共通部分の多さを表す値として，基準 4, 5 はその相関ルールから生成された雛形が開発者に与える情報量の多さを表す値として使用する．

表 2 データベース検索のためにソースコードから取得する識別子

識別子	内容
メソッド名の動詞	直前に記述したメソッド名の動詞
メソッド名の目的語	直前に記述したメソッド名の名詞節集合
クラス名	メソッド定義クラスの名前
親クラス名	メソッド定義クラスの親クラス名
インタフェース名	メソッド定義クラスのインタフェース名
フィールド名	定義されている全フィールドの名前

これらの基準から，以下の 4 つの変数を用意した．IDF は，学習に使用したソースコード集合から算出する．

B_r ： 支持度 (基準 1 より)
C_r ： 確信度 (基準 2 より)
N_r ： 条件部の要素数 (基準 3 より)
M_r ： 右辺の各呼び出しメソッドの IDF の総和 (基準 4, 5 より)

この各変数に重みはパラメータチューニングによって決定した $[0, 1]$ の範囲の定数 b, c, n, m を与え，以下の計算式に適用し，値が大きいものから順に候補を上から列挙する．この計算式は，基準となる各変数を足し合わせるものであり，得られる値 P_r を以後「優先度」と呼ぶこととする．

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

この 4 つの変数に与えるべき重みを決定するためのパラメータチューニングでは，相関ルールマイニングに使用したものとは別のソースコード集合を用意し，このソースコード集合のメソッド定義それぞれに対して本手法で雛形の候補を生成し，各メソッド定義の本体に記述されている呼び出しメソッドができるだけ上に並べられるような重みを求めることを考える．そこで，独自に評価値という値を定義し，この評価値ができるだけ高くなるような重みを求める．

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呼び出しメソッド集合である．

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

式中の $weight$ 関数は，IDF の値に基づいて各呼び出しメソッドに与える重みであり，以下の式が成り立つ．ここで $N(m)$ は，メソッド定義に記述されている呼び出しメソッドの数である．

$$weight(c) = \frac{IDF(c)}{N(m) \sum_{c \in C(m)} IDF(c)}$$

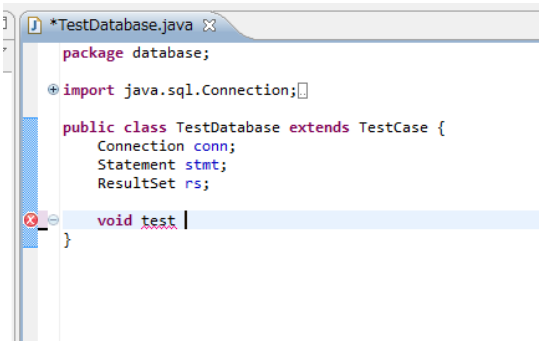

図 2　メソッド名を書いた状態でツールを起動する

式中の $maxRank(c, L(m))$ 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，全ての組み合わせで $E(M, L)$ が最も大きくなったパラメータの組み合わせを使用する．

4. ツールの実装

本ツールは Eclipse の Java エディタ内で動作し，ユーザが新規に作成するメソッドの名前を記述した状態で本ツールを起動すると，メソッド本体の雛形の候補リストが Eclipse のコードアシストを通して表示される．

例として，開発者が編集中の TestDatabase クラスを挙げる．開発者は，作成した TestDatabase クラスの中に必要なフィールドを宣言している状態であり，次に test というメソッドを新規作成したい．そこで返り値の型とメソッド名を記述した状態 (図 2) でツールを起動すると，test メソッドの本体で使用される可能性の高い雛形の候補リストが出現する (図 3)．開発者がこの雛形の候補リストから挿入したい候補を選択すると，雛形として各行が「呼び出しメソッドが定義されているクラスの完全修飾クラス名 => 呼び出しメソッド」の形式のコメントがメソッド本体に挿入される (図 4)．

5. 実験

本手法が，開発者に有用な API 集合を推薦できるかを評価するために，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 がわ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，開発者に手掛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

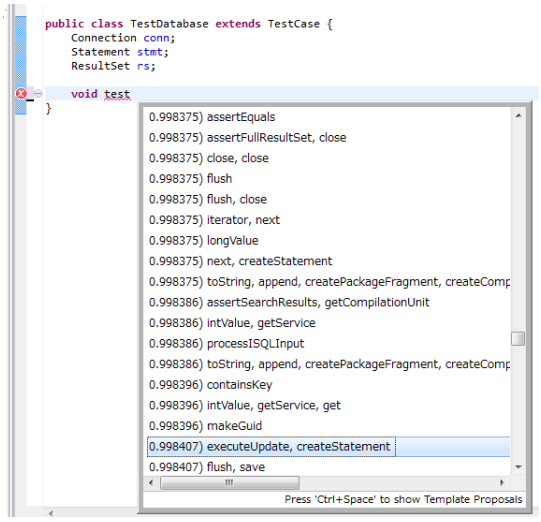

図 3　メソッド本体の雛形候補リストから挿入したい雛形を選択する

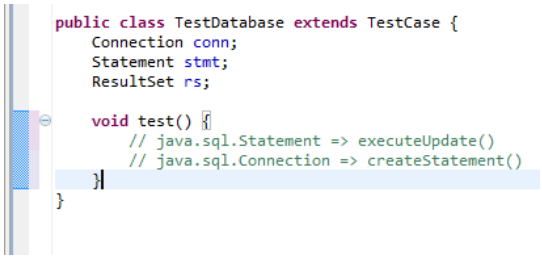

図 4　コメントの状態で挿入された雛形を修正してメソッド本体を完成させる

RQ2： ソースコードに記述されている識別子のうち，どの識別子が有用な推薦の手掛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

5.1 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 の手掛かりを与えられるか調査するために，メソッド本体と識別子の関連の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．

メソッド本体と識別子の関連の学習には，ソフトウェアプロダクトの収集・解析・検索システムである SPARS TYPE:R [6] の検索対象プログラムを使用した．これはプロジェクト数 443，ファイル数 198,324 であり，学習によって 5,401,676 の相関ルールを得られた．

本実験には，学習に使用したソースコード集合とは異なるオープンソースプロジェクトを使用した．これはプロジェクト数 517，ファイル数 310,166 であり，実験にはここからランダムで取り出した 10,000 ファイルを使用した．

5.1.1 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る呼び出しメソッドがどの程度提示できているかを調査した．これは，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，次にそのメソッドに対して本ツールで推薦を行い，最後に本ツールで得られた候補と削除したメソッド本体の比較を行うことで行った．

本 RQ では，既存のソースコードのメソッドに対して，そのメソッド本体に記述されている呼び出しメソッド (正解メソッド) が推薦できるかを適合率及び再現率で評価した．本手法では候補の並び替えについても提案しているため，有用な候補を上位に提示できているかを確認するために，適合率及び再現率を求める際，各順位までの候補におけるそれぞれの値を求めた．

適合率には 3 つの定義を用いた．これは，適合率を評価する際，正解の候補を定義する必要があるが，本手法では提示する候補が呼び出しメソッドの集合であるため，各候補が正しいかどうかを判定するのが難しいためである．適合率の定義は以下の 3 つである．

適合率 1： 提示した候補の中に，正解メソッドを 1 つでも含むメソッドがいくつ含まれていたかの割合

適合率 2： 提示した候補の全ての呼び出しメソッドの中に，正解メソッドがいくつ含まれていたかの割合

適合率 3： 提示した候補で同じ名前の呼び出しメソッドは最高順位のもの以外除去することで重複を除いた全呼び出しメソッドの中に，正解メソッドがいくつ含まれていたかの割合

再現率では，いくつの正解メソッドが候補に含まれていたかを評価する定義と，IDF による重み付けを行って再現率を評価する定義の 2 つを用いた．IDF による重み付けの評価は，開発者の誰もが知っているような頻繁に使われる呼び出しメソッドではなく，特定のライブラリで使用されるような呼び出しメソッドを提案できているか確認するために行った．ここで使用する IDF は，実験に用意した 517 個のオープンソースプロジェクトから算出した．再現率の定義は以下の 2 つである．

再現率： 全正解メソッド中，候補に含まれるものの割合

IDF に基づく再現率： 全正解メソッドの IDF の総和中，候補に含まれるメソッドの IDF の総和の割合

1 位，5 位，10 位，30 位までの候補と，候補全体における適合率 1, 2, 3，再現率，IDF に基づく再現率を表 3 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 4 に示す

5.1.2 RQ2： ソースコードに記述されている識別子のうち，どの識別子が有用な推薦の手掛かりとして使用されたか

推薦された候補は，そのソースコードに記述されている様々な情報を上手く活用して推薦されているかを調査するために，本手法で推薦した候補の中でメソッド本体と一致する候補の元となった相関ルールを調査し，どの識別子を条件部に持つ相関ルールが多いかを調査した．

その結果として，正解を含む候補の元となった相関ルールにおける，条件部の識別子の種類の割合を表 5 に示す．これは，各相関ルールの条件部に 1 つでもその種類の識別子が含まれていた場合 1 と数えており，各条件部は複数の識別子からなるため，割合の合計は 100%にならない．

5.1.3 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

コード片を推薦する既存手法として，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ものがいくつか提案されている．そこで，それらの手法と連携できるかを調査するために，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する．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に対して推薦を行い，候補全体の中に正解メソッドの前半分 (小数点以下切り捨て) 全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．

表 3 一部の順位と候補全体における適合率・再現率

	適合率 1	適合率 2	適合率 3	再現率	IDF 再現率
1 位	0.128	0.133	0.133	0.031	0.019
5 位	0.109	0.102	0.074	0.078	0.044
10 位	0.096	0.089	0.058	0.103	0.058
30 位	0.082	0.068	0.036	0.148	0.083
全体	0.053	0.027	0.008	0.290	0.183

表 4 一部の順位と候補全体における F 値

	適合率 1	適合率 2	適合率 3
1 位	0.050	0.050	0.050
5 位	0.091	0.088	0.076
10 位	0.100	0.096	0.075
30 位	0.106	0.093	0.058
全体	0.090	0.049	0.011

表 5 正解を含む相関ルールにおける，条件部の識別子の種類の割合

識別子	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%

その結果，メソッド定義のうち，メソッド本体に記述されている呼び出しメソッドが 2 つ以上のものは 85,389 個，そのうち前半部分の呼び出しメソッド全てがツールの提示した候補に含まれていたものは 6,945 個あり，約 8.1% のメソッド定義で前半の呼び出しメソッドを推薦できたことがわかった．

5.2 考察

表 3 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，適合率の値が減少する傾向が見られる．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ためであり，本手法では並び替えによって適切な候補を上位に並べられていると考えられる．本手法はメソッド本体が記述されていない状態で推薦を行っており，比較対象となる既存研究が存在しないため，表 3 や表 4 の値が良いか悪いかを評価することは難しい．また，学習用のソースコード集合に記述されていない API を推薦することは本手法では不可能である．推薦に利用する情報が少ないこと，学習と異なるソースコード集合を用いて評価を行ったことを考えると，約 29 % の再現率は決して低い値ではないと考えられる．以上から，RQ1 については，推薦された API 集合にはメソッド本体で使用されそうな API 集合が含まれていたと言える．

表 5 の結果を見ると，有用な推薦の手掛かりに使用された識別子にはバラつきがある．その中でメソッド名の動詞が 84.3% と特に大きいため，RQ2 については，有用な API の推薦にはメソッドの動詞が大きな手掛かりになることがわかった．また，メソッドの動詞以外の識別子の割合が小さいため，それらを上手く手掛かりとすることで，開発者の記述しているソースコードに，より適した API 集合を推薦することが期待できる．例えば，本手法ではクラス名やインタフェース名において名前全体が一致する相関ルールしか取得できなかったが，これを単語ごとに分割して相関ルールマイニングを行い，クラス名が部分一致する相関ルールを取得できれば，記述中のコードと似たクラス名から得られた API 利用実績でも推薦が行えるようになる．

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，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には，手法の改善が必要である．

最後に，本手法では，開発者がメソッド本体を記述する手掛かりとなる API 集合を推薦することを目指しているため，開発者が作成した呼び出しメソッドや，ソースコード中に定義されたメソッドと同名の API を推薦することは，ソースコード中に出現しない名前の API を推薦することに比べて重要度が低いと考えている．また，開発者が記述したメソッド定義の中でもプライベートメソッドについては，開発者自身が外に公開することを想定していないメソッドであり，処理内容や命名規則にふさわしい名前が付けられていない可能性が高いため，本ツールによって推薦する価値は少ないと考えられる．そこで，そのような重要度が低いと考えられる呼び出しメソッドが，評価に使用したソースコード中にどの程度含まれているかを調べた．すると，評価で使用したソースコード集合に記述されている呼び出しメソッド 344,250 個中，全体の約 18%を占める 61,754 個は同一ファイル内で定義されているメソッドであり，全体の約 5%を占める 17,007 個は同一ファイル内で定義されているプライベートメソッドであった．今回の実験では，開発者が実際に推薦されて嬉しい API を定義することが難しいため，既存のソースコードに記述されている呼び出しメソッドを全て正解メソッドと定義したが，正解メソッドの約 18 % が同一ファイルで定義されているメソッドだったことを考慮すると，実際に開発者にとって推薦してほしい API を正解メソッドと定義した場合の再現率は実際の結果より高いものであると考えられる．

6. 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ている [11] ．この研究では，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，API を推薦する手法を実装した APIExample [9] というツールを提案している．この手法では，ウェブベースのツール上に入力されたキーワードに応じて Java API を推薦するものである．Wang らの手法はウェブベースのツール上に推薦結果が表示されるのに対し，本手法で作成されたツールは統合開発環境のコード補完機能として動作するため，推薦結果を記述中のソースコードに直接挿入できる．また，Wang らの手法の情報源がインターネットのウェブページであることに対し，本研究では既存のソフトウェアである点も異なる．

Martin 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 [8] 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 [10] や 島田ら [12] 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．

7. まとめと今後の課題

本研究では，開発を行う際，膨大な API の中から必要な API を選択し，それをどのように組み合わせるかを考えるのが難しいという問題において，手掛かりや支援が少ない API の選択に着目し，開発者が新規作成したいメソッド名を記述したときに，そのメソッド本体で使用されるであろう API を推薦することで，API の選択を支援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調査したところ，推薦の際に並び替えによって適切な候補を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わかった．

今後の課題としては，API 集合ではなく API 列を開発者に提示することで，API を選択する作業だけでなく API を適切に組み合わせる作業の支援を検討している．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，日本学術振興会科研費基盤 (S) (課題番号 25220003)，日本学術振興会研究費補助金若手研究 (B) (課題番号 25730036) の助成を得た．

参考文献

[1] : OpenNLP, http://opennlp.apache.org/.
[2] : WordNet, http://wordnet.princeton.edu/.
[3] Agrawal, R., Imieliński, T. and Swami, A.: Mining Association Rules Between Sets of Items in Large Databases, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207–216 (1993).
[4] Bruch, M., Monperrus, M. and Mezini, M.: Learning from Examples to Improve Code Completion Systems, *Proceedings of the 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).
[5] Høst, E. W. and Østvold, B. M.: Debugging Method Names, *Proceedings of the 23rd European Conference on Object-Oriented Programming (ECOOP 2009)*, pp. 294–317 (2009).
[6] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations., *IEEE Transactions on Software Engineering*, Vol. 31, No. 3, pp. 213–225 (2005).
[7] Kashiwabara, Y., Onizuka, Y., Ishio, T., Hayase, Y., Yamamoto, T. and Inoue, K.: Recommending Verbs for Rename Method using Association Rule Mining, *Proceedings of IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)* (2014).
[8] Monperrus, M., Bruch, M. and Mezini, M.: Detecting Missing Method Calls in Object-Oriented Software, *Proceedings of the 24th European Conference on Object-Oriented Programming*, pp. 2–25 (2010).
[9] 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., *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pp. 592–595 (2011).
[10] Ye, Y., Fischer, G. and Reeves, B.: Integrating Active Information Delivery and Reuse Repository Systems, *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-first Century Applications*, pp. 60–68 (2000).
[11] 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).
[12] 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).