修士学位論文

題目

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

指導教員

井上 克郎 教授

報告者

鬼塚 勇弥

平成 26 年 2 月 5 日

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 ソフトウェア工学講座

平成 25 年度 修士学位論文

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

鬼塚 勇弥

内容梗概

開発者はソフトウェア開発で API を使用する際，大量の API 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する．開発者は推薦された API を参考にして編集を行いメソッド本体を完成させる．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．

主な用語

API
メソッド本体
相関ルールマイニング
コード補完

目 次

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．

多くのソフトウェアが共通して利用する機能はライブラリやフレームワークなどによって提供されており，これらが持つ機能は呼び出し手順や記述方法を定めた Application Programming Interface (API) を通じて提供されている．この API には，汎用性の高いプログラムの再開発を防ぐために使用するものや，ファイルの読み書きといった OS のシステムコールを用いるために使用するものなど，様々なものが存在する．ソフトウェアは複数の機能を組み合わせて構成されており，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を組み合わせる必要がある．このように，ソフトウェア開発では複数のライブラリやフレームワークを利用するため，開発者は多種多様な API を組み合わせて開発を行わなければならない．

しかし，多種多様な API の中から利用すべき API を選択し，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，直前に書かれたメソッド呼び出しの列から，次に書かれそうなコードの候補を推薦する手法がいくつか提案されている [11, 23] ．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 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，少なくとも数個の API を使用することを開発者が決め，メソッド呼び出し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，開発経験の少ない人や開発するソフトウェアの分野に精通していない人など，使用する数個の API を決めら

れない人には利用が難しい．

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 [14, 22] 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 の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．

提案手法を Eclipse [3] のコード補完機能を拡張することで実装した．本ツールは Java で記述されたプログラムを対象としており，Eclipse の Java エディタにおいてメソッド名を記述した後にコード補完機能を呼び出すと，本手法による API 集合から生成されたメソッド本体の雛形が推薦される．開発者はその候補から自分が使用したいものを選択し，メソッド本体に挿入して使用する．

以後，2 章では，本研究の背景となった概念に関して述べる．3 章では，本研究で取り組む問題について述べる．4 章では，API 集合を推薦する手法について説明する．5 章では，既存のソースコードの学習で得られたデータベースの詳細や，実装したツールの機能について述べる． 6 章では，実験とその考察を述べる．7 章では，本研究と関連がある研究について述べる． 8 章では，本研究のまとめと今後の課題に関して述べる．

2 背景

本研究の提案手法の背景として，まず本手法で推薦する API について説明する．次に，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，オブジェクト指向プログラムの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た後，学習に利用する相関ルールマイニング技術と，学習結果を格納するデータベースに使用する FP-Tree について説明する．最後に，学習した API 利用実績に基づく推薦ツールを実装する環境として，本研究で使用した Eclipse のコード補完機能について説明する．

2.1 Application Programming Interface (API)

Application Programming Interface (以下 API) とは，ソフトウェアコンポーネントが持つ機能を利用するためのインターフェイスの仕様である．多くのソフトウェアで利用される機能を，各々の開発者が毎回ゼロから開発するのは困難かつ無駄が多い．そこで，そのような機能は OS やミドルウェア，ライブラリといった形でまとめて提供されており，その機能を使用するためのインターフェイスが API である．API はプログラミング言語の関数やメソッドとして提供されることが多い．

API を用いてアクセスするソフトウェアコンポーネントには，言語の標準ライブラリ，商用オフザシェルフ (COTS)，開発環境などのプラットフォームが提供する機能，オープンソースのライブラリやソフトウェアフレームワークなど，無料のものから有料のものまで様々な種類が存在する．そのため，現在では API の数は膨大なものとなっている．既に他人が作った機能を再利用して効率的なソフトウェア開発を行うためには，これらの膨大で多種多様な API から必要なものを選び出して組み合わせる必要がある．

例として SQL を扱う Web アプリケーションを考える．Web アプリケーションとして機能するためには，まずクライアントから送られてきたリクエストに含まれている情報を取得する．次に，特殊文字などが含まれた文字列をプログラム内で SQL クエリとして直接実行することができないため，エスケープという文字列処理を行う．そして最後に，エスケープされた SQL クエリでデータベースを更新する．この例ではクライアントのリクエストを扱うライブラリ，文字列処理を扱うライブラリ，そして使用するデータベースのライブラリという 3 種類のライブラリが必要となる．このように，ソフトウェア開発では複数のライブラリを利用するため，開発者は多種多様な API を組み合わせて開発を行わなければならない．

2.2 オブジェクト指向プログラムのメソッド名とメソッド本体の関係

本節では，オブジェクト指向プログラムのメソッド名には一般的にその処理内容を表す動詞が用いられることの説明として Java のメソッドの命名規則を例に挙げた後，そのメソッ

ド名中の動詞とメソッド本体に強い関係があるという事実について述べる．

2.2.1 Java のメソッドの命名規則

Java のメソッド名は，Oracle が公開している Code Conventions for the Java Programming Language [1] で示されているように，一般的には小文字で始まる動詞，もしくは 2 つ目以降の単語が大文字で始まる複数の単語で構成される．また，ここに書かれている以外にも，Java のメソッド名には様々な守られるべきルールが存在する．それらのルールの一部を以下に列挙する．

- フィールドに値を設定するメソッド (Setter) は，フィールド名の前に set を付け加えた名前とする．
- フィールドの値を返り値とするメソッド (Getter) は，フィールド名の前に get を付け加えた名前とする．
- boolean 型の値を返すメソッドは，is，can，has で始まる名前とする．

2.2.2 メソッド名とメソッド本体の関係

メソッドの命名に使われる動詞は，メソッド本体で呼び出しているメソッド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 [14] や柏原ら [22] 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，Instance という単語を使用した呼び出しメソッドが多く，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，呼び出しメソッド hasNext と iterator が同時に使用されていることが多い．

2.3 相関ルールマイニング

相関ルールとは，ある事象が発生すると別の事象が発生するといったような，同時性や関係性が強い事象間の関係のことであり，データベースに蓄積された大量のトランザクションからこの相関ルールを抽出する技術を相関ルールマイニングという．トランザクションとは一つの事象から得られるアイテムの集合のことであり，例えば商品の購買履歴であれば各商品がアイテムであり，一人の客が購入した商品の集合がトランザクションとなる．ある事象 X の下である事象 Y が発生する関係は矢印を用いて $X \Rightarrow Y$ と記述されることが多く，矢印の左側 (X) を条件部，矢印の右側 (Y) を帰結部という．相関ルールマイニングは，POS システムで収集された大量の購買履歴から同時に購入されやすい商品を導くときなどに利用されることで有名である [10, 16] .

相関ルールには，そのルールの重要度を表す指標として支持度と確信度という値がある．支持度は全トランザクションのうち，条件部と帰結部に登場する全てのアイテムを含むトランザクションの割合である．これはそのルールが登場する頻度を表わす．確信度は，条件部が含まれるトランザクションのうち，帰結部を含むトランザクションの割合である．

例として，表 1 の客 A 〜 D が購入した商品購買履歴について考える．この購買履歴の 牛乳 ⇒ パン という相関ルールに着目すると，購買履歴の客は 4 人，条件部である牛乳を購入した客は A, B, D の 3 人，牛乳とパンの両方を購入している客は B と D の 2 人なので，支持度は 2/4 ，確信度は 2/3 となる．

表 1: 商品購買履歴例

A さん	おにぎり	牛乳	ジャム
B さん	パン	牛乳	雑誌
C さん	おにぎり	お茶	
D さん	パン	牛乳	お茶

単純に全ての組み合わせを作って相関ルールを求めるのは実時間で計算するのが難しい．そこで相関ルールの抽出には一般的に Apriori というアルゴリズムが使われる [9] ．Apriori は，あるアイテム集合の支持度が，必ずそのアイテム集合の部分集合の支持度以下であることを使って計算を省略するアルゴリズムであり，利用者はある支持度未満の相関ルールを取得しないという最小支持度を設定して相関ルールを求める．このアルゴリズムによって，最小支持度の閾値次第では，実用的な時間内での相関ルール抽出が可能となる．

2.4 FP-Tree

FP-Tree [12] は，頻出アイテムをメモリ上に格納するための木データ構造である．これはトライ木のデータ構造と類似しており，トライ木は文字列の集合を格納するのに対し，FP-Tree ではトランザクション (アイテムの集合) の集合を格納する．FP-Tree は根に近いノードほどトランザクション集合での出現頻度が高いアイテムが格納される．

表 2 の左の列の 6 つのトランザクション集合を FP-Tree に格納する例を考える．トランザクション集合における各アイテムを出現回数の多い順 (表 3) に並べたのが表 2 の右の列のトランザクションである．このトランザクション集合を上から順に木構造に格納し FP-Tree を構築する手順の一部を図 1 に示した．根ノードである *root* のみの空の状態の FP-Tree にまずは $\{f, a, m, p\}$ を追加すると， *root* から出現頻度の多い順に各アイテムのノードが作成される．各ノードに記述されている数字はそのパスにおける各アイテムの出現回数である．次に $\{f, c, a\}$ を追加するときは，根から出現回数の多い順に辿ったとき $\{f\}$ は既にノードが

あるので出現回数を追加し，以降の $\{c, a\}$ は子ノードとして存在しないので新しくノードが作られる．更に $\{f, c, m\}$ を追加するときは，$\{f, c\}$ は既にノードがあるため出現回数だけ追加され，その子に $\{m\}$ がないので新しくノードが作られる．このようにして FP-Tree を構築していき，完成した FP-Tree が図 2 である．

表 2: トランザクション集合

トランザクション	出現頻度順に整列したトランザクション
$\{a, f, m, p\}$	$\{f, a, m, p\}$
$\{a, c, f\}$	$\{f, c, a\}$
$\{c, f, m\}$	$\{f, c, m\}$
$\{a, c\}$	$\{c, a\}$
$\{c, f\}$	$\{f, c\}$
$\{f\}$	$\{f\}$

表 3: 各アイテムの出現頻度

アイテム	出現回数
f	5
c	4
a	3
m	2
p	1

2.5 統合開発環境のコード補完機能

統合開発環境 (IDE) の多くはコード補完機能を有している．この機能は，編集中のソースコードからユーザがカーソル位置に記述するであろうコード片を推測してポップアップリストに表示し，ユーザが選択したコード片を挿入するというものである．コード補完で挿入されるコード片は，IDE に推測することが難しい部分を空白にした雛形の場合もある．雛形が挿入されたときは，開発者は空白部分に編集を加えて完全なコード片を完成させる．

コード補完を持つ統合開発環境は様々ある [2, 4, 6, 8] が，その一つとして Eclipse [3] を例に挙げる．Eclipse のコード補完機能はコードアシストと呼ばれており，プラグインとして新しい機能を組み込むことが容易に行えるように設計されている．Eclipse でソースコード編集中に Ctrl + Space を入力することにより，カーソル位置に記述されそうなコード片を補完することが可能である．挿入されるコード片は，定義されている変数・メソッドの名前や

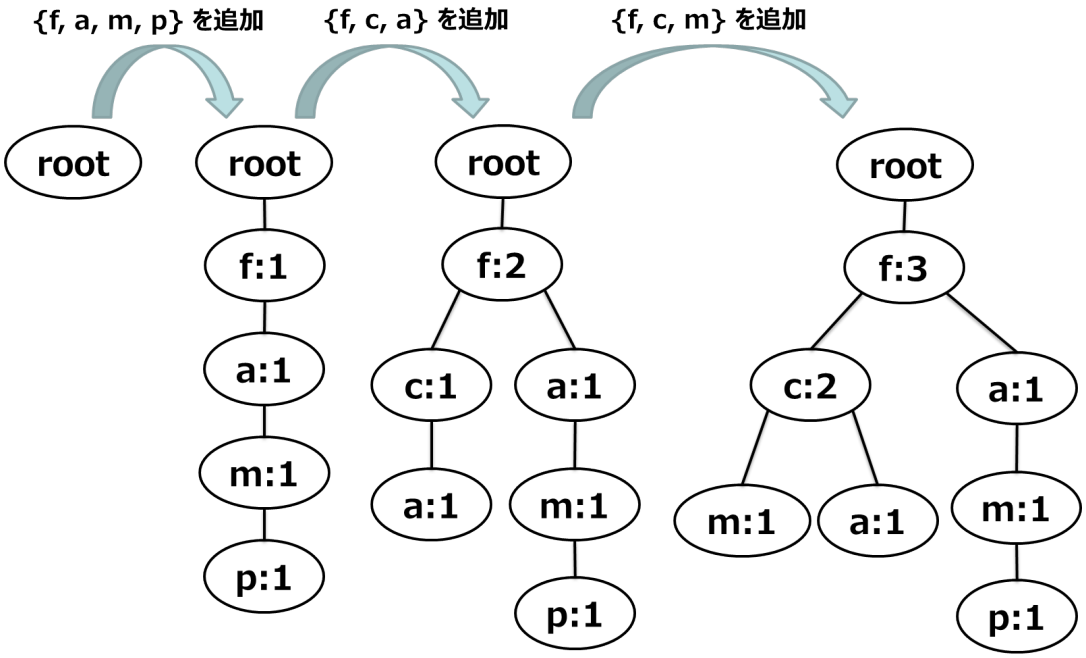

図 1: FP-Tree を構築する手順

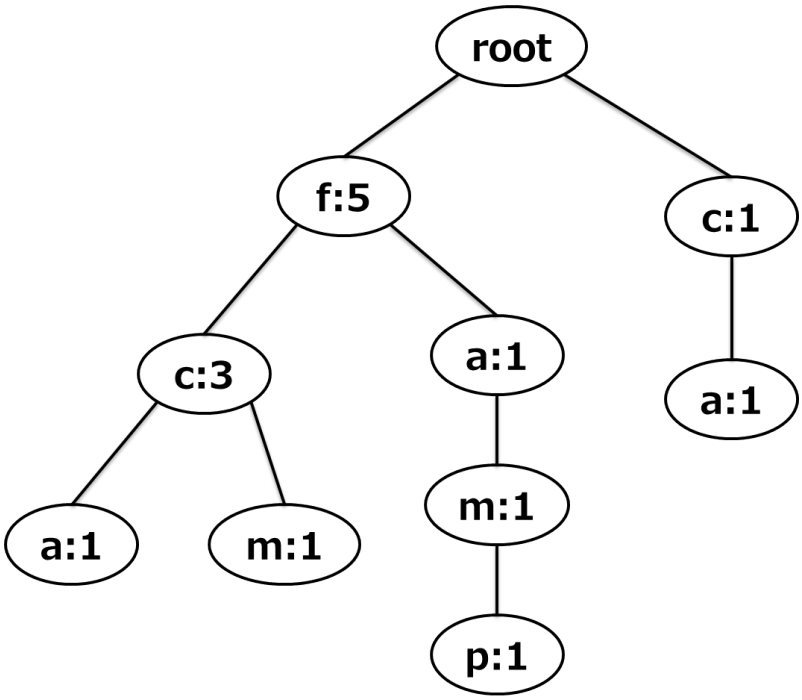

図 2: 完成した FP-Tree

分岐・ループといった単純な構文の他，クラスで宣言されている変数の値の取得・更新を行うメソッド定義などいくつかの種類がある．

3 問題の定義

開発を行う際，必要な API を選択し，それらを適切に組み合わせるのは難しいという問題がある．これは，API の数が膨大であり，それぞれの API の使い方が異なるためである．開発者が使用したい API を選択する際や，選択した API の使い方を知る際，ドキュメントやサンプルコード，API 自体のソースコードなどを調査する必要があるが，API の数が増加するにつれてその調査の労力も大きくなっている．

効率的なソフトウェア開発を行うために，直前に書かれたメソッド呼び出しの列から，次に書かれそうなコードの候補を推薦する手法がいくつか提案されている [11, 23] が，少なくとも数個の API を使用することを開発者が決め，メソッド呼び出し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，新規作成するメソッドには使用できない．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 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が，開発経験の少ない人や開発するソフトウェアの分野に精通していない人など，使用する数個の API を決めることも難しい人には依然として API の利用が難しいものとなっている．

API の使用には，必要な API を選択する作業と，選択した API を適切に組み合わせる作業の 2 つの作業が必要であるが，本研究では API の選択について着目する．これは，API を組み合わせる作業に比べて，API を選択する作業は手掛かりが少なく難しいためである．また，API を組み合わせる作業には API 利用方法推薦システム [19] などによる支援も多い．そのため，開発者に手掛かりを与え API の選択を支援できれば，知識に乏しい開発者が API を使用するために必要な調査の手間を縮めることが期待できる．

4 提案手法

本研究では，膨大な API から必要なもの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 の集合を雛形として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．開発者は推薦された雛形からコードに記述したいものを選択・挿入し，挿入された雛形を並び替えたり足りないコードを書き加えたり，推薦された雛形をインターネット検索や API 利用方法推薦システムなどの手掛りとして使用し，メソッド本体を完成させる．

本研究では， API として呼び出しメソッドを推薦し，その集合をメソッド本体の雛形と呼ぶ．呼び出しメソッドとは，メソッド本体から呼び出して使用しているメソッドのことである．呼び出しメソッドを推薦するのは，メソッド本体がメソッド呼び出し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 [15, 19] で調べられることに基づく．提案手法を用いることで，プログラミング初心者が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とき，本手法で提供される雛形はメソッド本体の記述や必要な情報の検索の手掛りになる．

本研究では，ソースコードを記述中の開発者に対して，メソッド本体で使用する可能性の高い API 集合を推薦するという戦略によって，API の選択を支援する．これは，開発者がキーワードを入力して API を検索するシステムと異なり，開発者が作業を中断する必要がなく，記述中のソースコードの情報をもとに推薦を行うため開発者自身が API を検索するキーワードを考える必要がないという利点がある．

メソッド本体で使用する可能性の高い API 集合を推薦するためには，過去の API 利用実績から記述中のソースコードに適したものを取り出し，開発者に提供する必要がある．過去の API 利用実績は，メソッド名とメソッド本体に強い関連があること [14, 22] 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，a) メソッド本体と識別子の関連の学習，b) メソッド本体の雛形の推薦 の 2 ステップからなる (図 3)．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．メソッド本体と識別子の関連の学習は，既存のソフトウェアから API の利用実績を学習するステップである．大規模なソフトウェア集合から生成したトランザクション集合で相関ルールマイニングを行い，抽出した相関ルールをデー

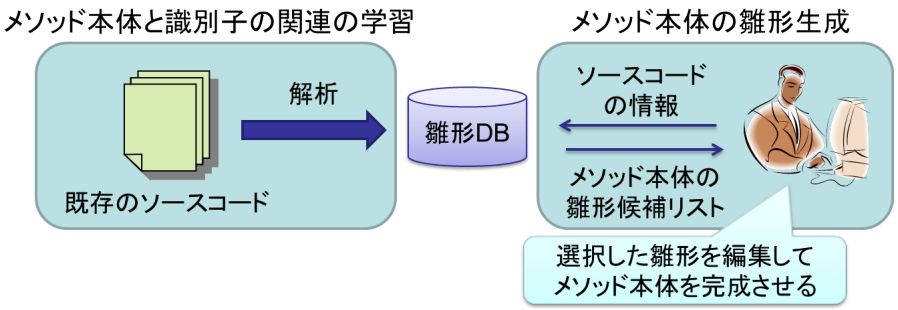

図 3: 提案手法の概要

タベースに格納して雛形 DB を作成する．メソッド本体の雛形の推薦は，ソースコードを記述中の開発者にメソッド本体の雛形の候補を提示するステップである．記述中のソースコードから取得したアイテム集合で 雛形 DB を検索し，得られた相関ルールからメソッド本体の雛形を生成して推薦する．データベースの検索では，条件部の全アイテムが記述中のソースコードで得られた識別子に含まれるような相関ルール全てを高速に取得する必要がある．そこで，本手法では FP-Tree に基づく部分集合検索アルゴリズムを利用して検索を行う．

推薦する雛形の候補リストは，メソッド本体で書かれそうな順に並び替えてから開発者に提示する．この並び替えは，筆者があらかじめ決めた基準に重みを与えて組み合わせた値に従って行った．この重みは，既存のソースコードでツールを使用したときに，定義されているメソッド名ができるだけ上に表示されるようにパラメータチューニングを行って決定した．

以降では，メソッド本体と識別子の関連の学習とメソッド本体の雛形の推薦のそれぞれの詳細を説明後，FP-Tree を用いた部分集合検索アルゴリズムとパラメータチューニングについて説明する．

4.1 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は図 4 のように，トランザクション集合の生成と相関ルールの抽出の 2 ステップからなる．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．

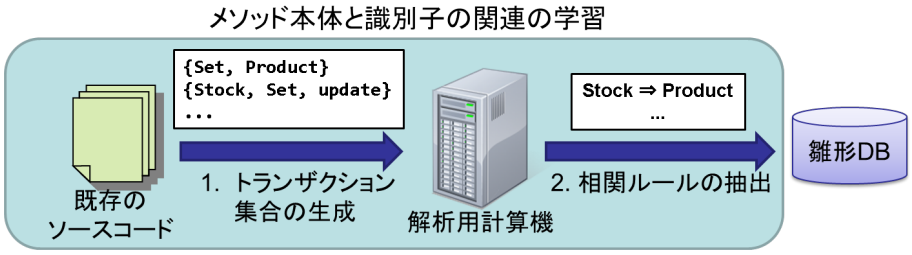

図 4: メソッド本体と識別子の関連の学習の概要

4.1.1 Step a-1. トランザクション集合の生成

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 4 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 4 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．

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．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．

表 4: 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	メソッドが定義されているクラスの名前
親クラス名	メソッドが定義されているクラスの親クラスの名前
インタフェース名	メソッドが定義されているクラスのインタフェースの名前
呼び出しメソッド名	メソッドの中で呼び出しているメソッドの名前
フィールド名	メソッド本体で使用している全てのフィールドの名前

4.1.2 Step a-2. 相関ルールの抽出

API の利用実績を学習するため，相関ルールマイニングを行い，トランザクション集合から相関ルールを抽出する．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．

4.2 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は図 5 のように，識別子の取得，データベースの検索，候補の提示の 3 ステップからなる．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．

4.2.1 Step b-1. 識別子の取得

雛形 DB の検索に用いる識別子を取得するため，開発者が記述中のソースコードに対して構文解析を行い，相関ルールを検索す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 5 に示す 6

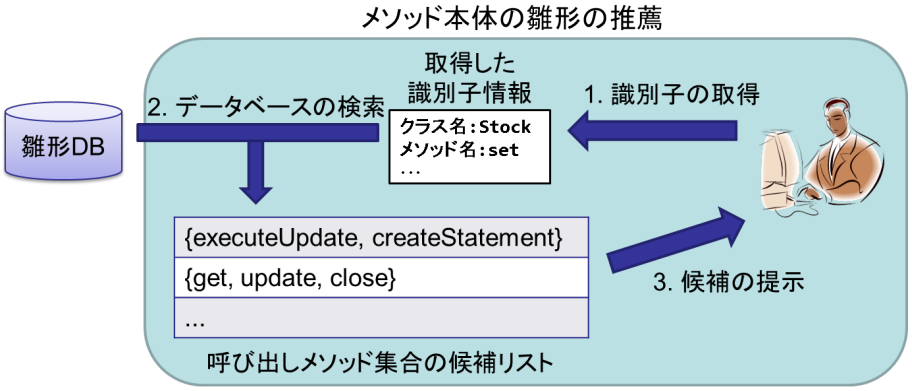

図 5: メソッド本体の雛形の推薦の概要

種であり，これは表 4 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．

表 5: データベース検索のためにソースコードから取得するアイテム

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に使われている動詞
メソッド名の目的語	直前に記述したメソッド名に使われている名詞節の集合
クラス名	メソッド名を記述したクラスの名前
親クラス名	メソッド名を記述したクラスの親クラスの名前
インタフェース名	メソッド名を記述したクラスのインタフェースの名前
フィールド	クラスで定義されている全てのフィールドの名前

4.2.2 Step b-2. データベースの検索

記述中のソースコードに関連の強い相関ルールを雛形 DB から検索する．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．出力する相関ルール 1 つは，(条件部，帰結部，支持度，確信度) の 4 種類の情報からなる．

ここで検索する相関ルールは，条件部の全要素が Step b-1 で得られた全アイテムに含まれるようなルールである．すなわち，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クエリとなるアイテム集合を $Q = \{q_1, q_2, \ldots, q_n\}$，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 を検索する．

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

大きなルール集合に対して短時間でこの検索を行うために FP-Tree を利用する．FP-Tree に基づく部分集合検索アルゴリズムの疑似コードをリスト 1 に示す．この疑似コードにおいて，car はリストの先頭の要素を取り出す関数，cdr はリストから先頭を取り除いたリストを返す関数，freq_rank は出現頻度の順位表から引数となるアイテムの出現頻度順位を取得する関数である．出現頻度の順位表では，同じ出現頻度に複数のアイテムが存在する場合それらは辞書順に順位付けられるため，異なる 2 つのアイテムに対して必ず出現頻度順位の上下を決定できる．

リスト 1: FP-Tree に基づく部分集合検索アルゴリズムの疑似コード

```
# node_list = 探索中の FP-Tree のノードのリスト
# sorted_query = 出現頻度順に並び換えられたクエリとなるアイテム集合

def find_subset(node_list, sorted_query)
  return [] if empty(node_list)
  return [] if empty(sorted_query)

  n_l_head = car(node_list)
  q_head = car(sorted_query)

  if n_l_head == q_head
    list = [[n_l_head]]
    children_subset = find_subset(n_l_head.children_list, cdr(sorted_query))
    rest = find_subset(cdr(node_list), cdr(sorted_query))
    return list + map(lambda(l){n_l_head + l}, children_subset) + rest
  elsif freq_rank(n_l_head) < freq_rank(q_head)
    return find_subset(cdr(node_list), sorted_query)
  elsif freq_rank(n_l_head) > freq_rank(q_head)
    return find_subset(node_list, cdr(sorted_query))
  end
end
```

FP-Tree に基づく部分集合検索アルゴリズムの概要は以下のとおりである．

1. ノードリスト N に根ノードを入れ，クエリとなるアイテム集合 Q を出現頻度でソートする
2. N の先頭を n，N から n を除いた集合を N’，Q の先頭を q，Q から q を除いた集合を Q’ とし，N か Q が空になるまで以下を再帰的に繰り返して部分集合を得る
   (a) n と q が一致したら以下の 3 つを部分集合に追加する
      i. n のみからなる集合
      ii. n の子ノード集合と Q’ から再帰的に検索して得られた集合それぞれに n を加えたもの
      iii. N’ と Q’ から再帰的に検索して得られた集合

(b) n の出現頻度順位が q の出現頻度順位より上であれば，N' と Q から再帰的に検索して得られた集合を部分集合に追加する

(c) n の出現頻度順位が q の出現頻度順位より下であれば，N と Q' から再帰的に検索して得られた集合を部分集合に追加する

4.2.3 Step b-3. 候補の提示

開発者にメソッド本体の雛形リストを提示する．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．並び換えのために筆者が考えた基準は以下の 5 つである．

基準 1： 支持度が高い

基準 2： 確信度が高い

基準 3： 左辺の要素数が多い

基準 4： 右辺の要素数が多い

基準 5： 右辺に出現するアイテムの希少度が高い

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．

これらの基準から，以下の 4 つの変数を用意した．IDF とは対象とするもの (ここでは呼び出しメソッド) の希少度であり，その呼び出しメソッドがどれだけ特徴的であるかを値で表現したものである．

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)

この各変数に重みとして，4.3 節で後述する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．

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

4.3 パラメータチューニング

4.2.3 で並び換えの基準に応じた 4 つの変数を用意したが，良い並び換えを行うためにはそれぞれの変数に対してどの程度の重みを与えるべきかを調べる必要がある．そこで，並べ替え優先度の計算に用いる各変数の重みを決定するためにパラメータチューニングを行った．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た．

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．

$$\begin{aligned} weight(c) &= \frac{IDF(c)}{N(m) \sum_{c \in C(m)} IDF(c)} \\ \sum_{c \in C(m)} weight(c) &= 1 \end{aligned}$$

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．$maxRank(c, L(m))$ はリストの上位から順に見ていき，c を含む候補を発見したらそのリストの順位を出力する．

パラメータチューニングは2段階で行った．まず，各パラメータに対して $0.8^0, 0.8^1, 0.8^2, ..., 0.8^{20}$ の 21 個の値を用意し，それらのパラメータの全組み合わせの中で $E(M, L)$ が最も大きくなるパラメータの組み合わせを調べた．次に，$E(M, L)$ が最も大きくなる組み合わせの中で，一番大きなパラメータを 1 に固定し，それ以外のパラメータについては $0.8^0, 0.8^1, 0.8^2, ..., 0.8^{40}$ の 41 通りの値を用意して，$E(M, L)$ が最も大きくなるパラメータの組み合わせを調べた．2回目の $E(M, L)$ が最も大きくなるパラメータの組み合わせをパラメータとして利用した．

5　ツールの実装

前節で述べた提案手法を Eclipse で使用するツールとして実装した．Eclipse は Java で記述したプラグインによって新しい機能を組み込むことが容易に行えるという特徴がある．そこで，本ツールは Eclipse のコードアシストを拡張することで作成した．対象はオブジェクト指向言語の Java のソースコードとした．本節では実装したツールの機能や使用方法，相関ルールを格納するデータベースの詳細について説明する．

5.1　メソッド本体と識別子の関連学習機能

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集合に対して Java ソースコードの構文解析や品詞解析を行う必要がある．本ツールを Eclipse を拡張して作成して作成していることから構文解析には 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．また，メソッド名の動詞と目的語の特定のために行った品詞解析には，Java で実装されたオープンソースソフトウェアプロジェクトの一つである OpenNLP [5] と，無料で利用でき Java プログラムから使用するためのライブラリも提供されている辞書 WordNet [7] を使用した．

相関ルールマイニングの結果はデータベースに格納するが，実装したツールでは Key-Value ストアの Berkeley DB を使用した．Key-Value ストアのデータベースの Key から Value の内容を引く操作が高速に行えるという特徴が FP-Tree の検索アルゴリズムに適しており，Key-Value ストアのデータベースの一つ Berkeley DB は Java で実装された Java Edition がオープンソースとして公開されているため，本ツールで使用するデータベースとして Berkeley DB を選択した．Key-Value ストアのデータベースに FP-Tree を構築する方法は 5.3 章で後述する．

5.2　雛形の推薦機能

雛形の推薦では，データベースを検索するアイテムとしてソースコード中に記述したメソッド名や周りの識別子を利用するため，関連の学習と同様に Java ファイルの構文解析や品詞解析を行う必要がある．これについては，関連の学習と同様の手順で行った．また，データベースでは取得したアイテムの部分集合を条件部に持つ相関ルールを検索するが，その詳細については後述する．

パラメータチューニングでは，パラメータを変更する度に対象のソースコード集合の全メソッド定義に推薦を行うと，データベースの検索を頻繁に行うためチューニングが完了するのにかなりの時間を要するという問題がある．そこでパラメータチューニングでは，パラメータを変更したときに候補の順番は入れ替わっても，候補の数や内容に変化がないことを

利用し，パラメータチューニングに使用したソースコードのメソット定義全てに対する推薦結果を先にメモリに書き出し，パラメータを変更するだけで並び換えが行えるように工夫した．これにより，データベースへのアクセスが初回の推薦結果取得時のみとなり，パラメータチューニングを高速に行うことができた．

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．ツールの起動は，カーソルがメソッド名の後ろにある状態で，コードアシスト同様 Ctrl + Space キーを入力することで行われる．

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．そこでアクセス修飾子とメソッド名を記述した状態 (図 6) で Ctrl + Space を入力してツールを起動する．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 7)．開発者はこの雛形の候補リストから挿入したい雛形を方向キーやマウスで選択することができ，Enter キーを押す (マウスをクリックする) とコメントの状態の雛形がメソッド本体に挿入される (図 8)．挿入されるコメントは，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」の形式で挿入される．開発者はこの挿入されたコメントを編集してメソッド本体を完成させる．なお，メソッド本体の雛形候補の先頭に表示されている数値は，並び替えで生成した優先度の最大値から各雛形候補に対応する優先度を引いた値である．コードアシストの内容は辞書順に並べられるため，優先度の最大値から優先度を引いた値が小さいほど，すなわち優先度が大きいほど上位に表示される．

5.3 データベースの構築

本手法では，FP-Tree の形式で相関ルールをデータベースに格納する．後述する部分集合検索に適した FP-Tree を構築するため，条件部を FP-Tree のトランザクションとし，出現頻度順に整列したトランザクションの最後のアイテムノードに相関ルールを格納する．これにより，任意のノードに格納された相関ルールの条件部はそのノードから根までのパスで表現できる．例えば，「$\{f, a, m, p\} \Rightarrow \{x, y, z\}$, 支持度 = 0.6, 確信度 = 1.0」という相関ルールによって，根ノードが f，葉ノードが p の FP-Tree が生成された場合，葉ノード p には出現回数の代わりに「確信度 = $\{x, y, z\}$, 支持度 = 0.6, 確信度 = 1.0」という情報が格納される．

本ツールでは，相関ルール集合を格納した FP-Tree を Key-Value ストアの Berkeley DB 上に構築した．データベース内部では，各アイテムに ID を割り当てる ITEM_DB，FP-Tree の木構造を格納する TREE_DB と，ルールを格納する RULE_DB に分かれる．

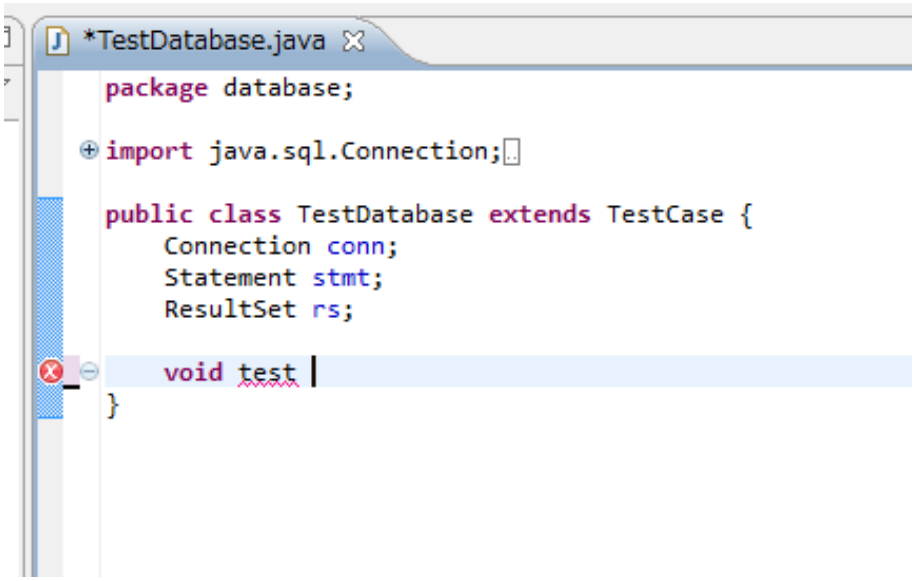

図 6: メソッド名を書いた状態でツールを起動する

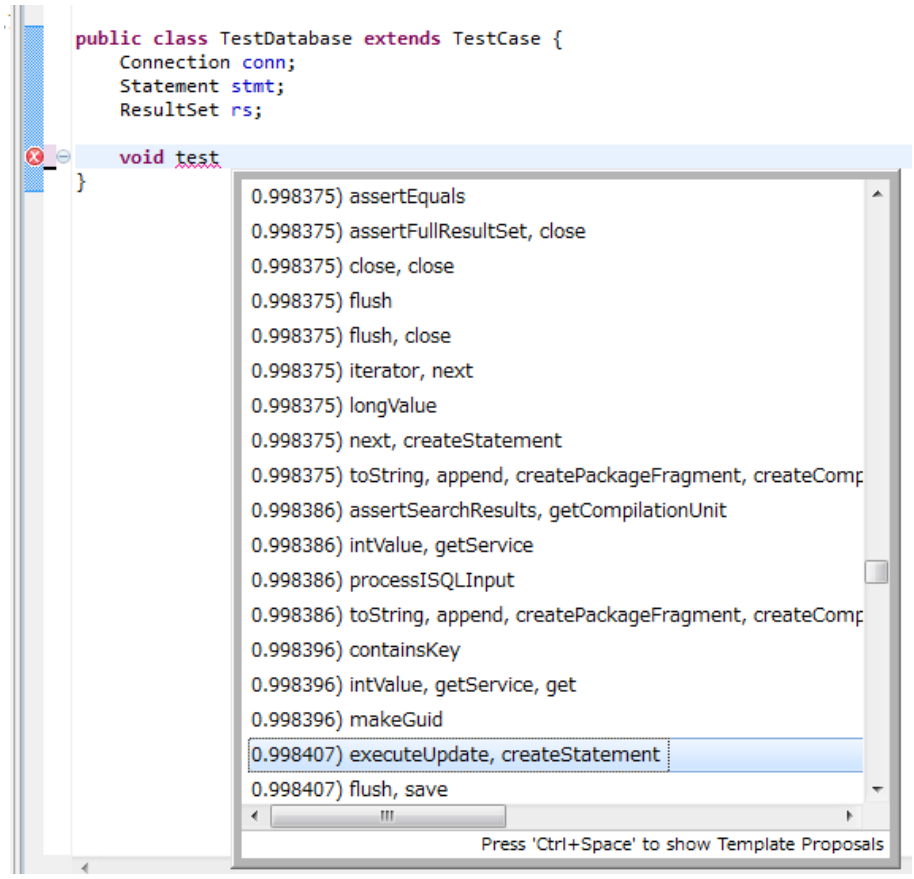

図 7: メソッド本体の雛形候補リストから挿入したい雛形を選択する

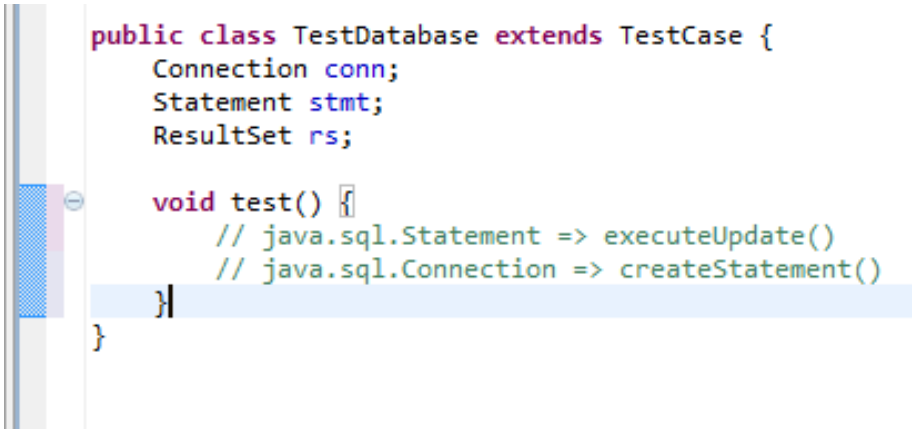

図 8: コメントの状態で挿入された雛形を修正してメソッド本体を完成させる

ITEM_DB はアイテムとなる各識別子の文字列のそれぞれに ID を割り当てており，Key がアイテム ID，Value が文字列となっている．ITEM_DB では出現頻度の高い順に ID を割り当てることにより，FP-Tree 検索においてアイテム ID の比較だけで出現頻度を比べることができるように工夫している．

TREE_DB は全ノードの親子関係を格納しており，Key がノード ID，Value が (子ノードのアイテム ID, 子ノードの ID) の組のリストとなっている．FP-Tree の根ノードの ID が Key であるような Key-Value ペアを検索しその Value から子ノードの ID を取得，更に取得した ID が Key であるような Key-Value ペアを検索しその Value から更に子ノードを取得する，という操作を繰り返すことで FP-Tree の根ノードから葉ノードまでを辿ることができる．

RULE_DB には全ルールを管理しており，Key がルール ID，Value がそのルール ID を割り当てられたルール 1 つとなっている．TREE_DB の Value に格納されているルール ID で RULE_DB を検索することにより，そのノードに格納されたルールを取得することができる．

6 実験

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，開発者の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： ソースコードに記述されている様々な情報を上手く活用して API 集合を推薦できているか

RQ3： 直前に書かれたメソッド呼び出し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連携できるか

6.1 方法

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 [15] の検索対象プログラムを使用した．これは 443 個の Java プロジェクトであり，合計で 198,324 ファイルのソースコードが含まれている．このソースコード集合から得られた識別子に対して，1 つのルール中に含まれる全アイテムが少なくとも 40 以上のトランザクションで出現するという閾値を支持度に設けて相関ルールマイニングを行ったところ，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 517 個の Java プロジェクトであり，合計で 310,166 ファイルのソースコードが含まれている．このソースコード集合からランダムで 10,000 ファイルを取り出し，全てのリサーチクエスチョンに対してこれを使用して評価を行った．この 10,000 ファイルには 85,392 個のメソッド定義が記述されており，そのうちメソッド本体に 1 つ以上呼び出しメソッドが記述されているメソッド定義は全部で 51,873 個であった．本実験ではこの 51,873 個のメソッド定義を対象に行った．

6.1.1 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

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に用意した評価用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．評価対象のメソッド定義は，メソッド本体に 1 つ以上呼び出しメソッドが記述されているものとした．

本リサーチクエスチョンでは，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 (以下，これを正解メソッドと呼ぶ) が推薦できるかを適合率及び再現率で評価する．本手法では候補の並び換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，各順位までの候補における適合率及び再現率を求めた．また，本手法で推薦する候補は呼び出しメソッドの集合であるため，正解メソッドとそうでないメソッドの両方を含む候補が出現し，各候補が正しいかどうかを判定するのが難しい．そこで，適合率に関しては 3 つの定義を用意し，それぞれについて調査を行った．また，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，再現率については IDF による重み付けを行った定義も用意した．この 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．適合率及び再現率の定義は以下の通りである．

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： 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

6.1.2 RQ2： 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 API 集合を推薦できているか

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．

6.1.3 RQ3： 直前に書かれたメソッド呼び出し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連携できるか

コード片を推薦する既存手法として，直前に書かれたメソッド呼び出しの列から，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，候補全体のうち正解メソッドの前半分 (小数点以下切り捨て) 全てが推薦できたメソッド定義はどの程度あるかを調査した．

6.2 結果

6.2.1 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

50 位以上の候補に対する適合率 1, 2, 3 をそれぞれ図 9, 10, 11 に，再現率を図 12 に，IDF に基づく再現率を図 13 に示す．また，1 位，5 位，10 位，30 位，候補全体における適合率，再現率の値を表 6 に示す．

表 6 の結果から F 値の計算も行った．図 14, 15, 16 はそれぞれ適合率 1, 2, 3 と再現率から計算した F 値である．1 位，5 位，10 位，30 位，候補全体における正確な適合率，再現率の値は表 7 に示す．F 値が最も高いのは，適合率 1 のときが 67 位，適合率 2 のときが 18 位，適合率 3 のときが 5 位以上の候補を対象としたときであった．

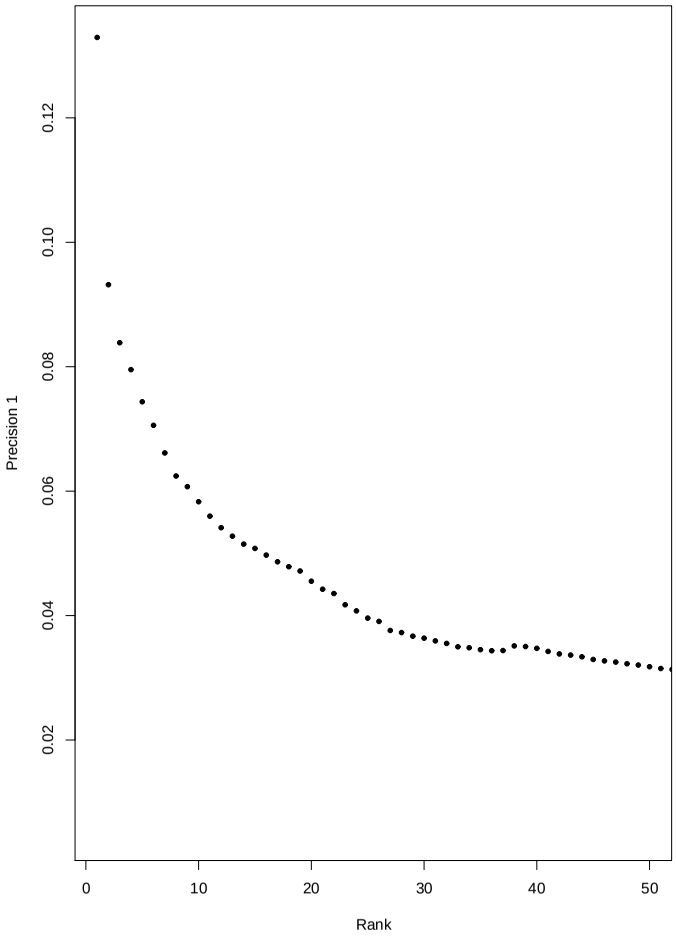

図 9: 50 位以上の候補に対する適合率 1

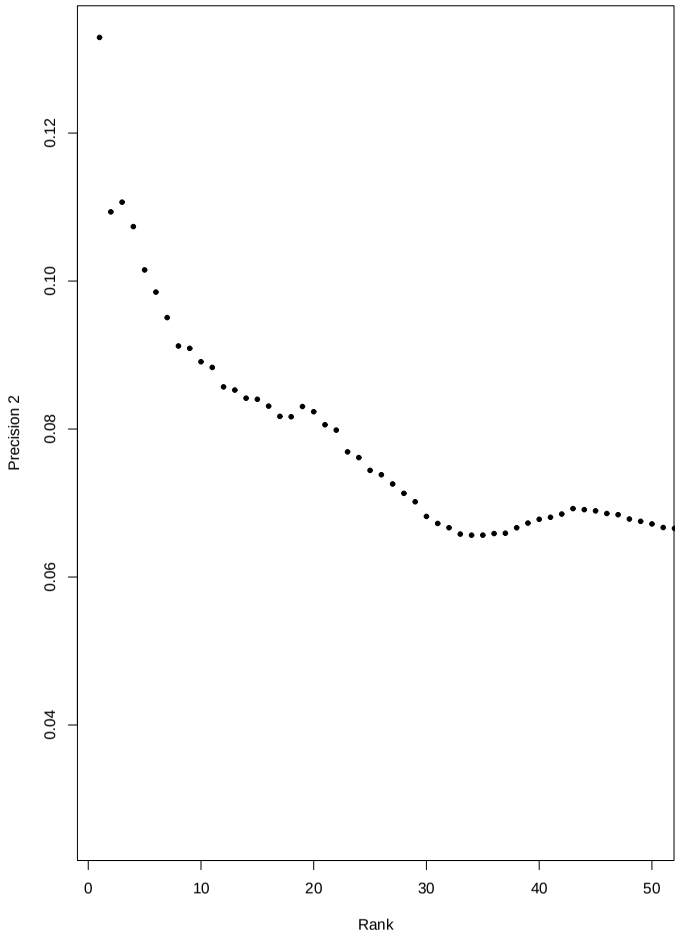

図 10: 50 位以上の候補に対する適合率 2

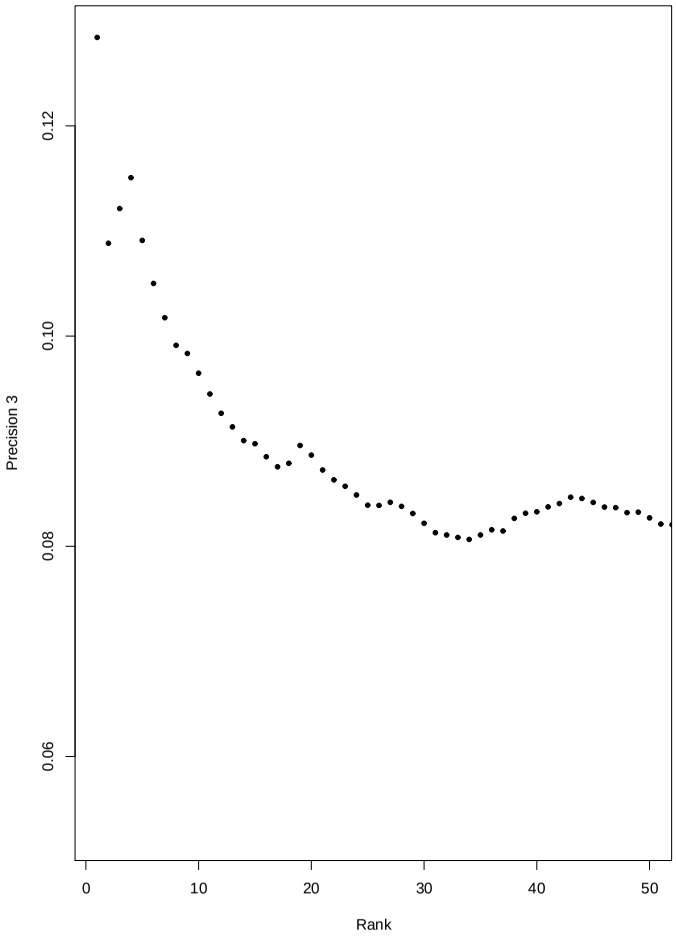

図 11: 50 位以上の候補に対する適合率 3

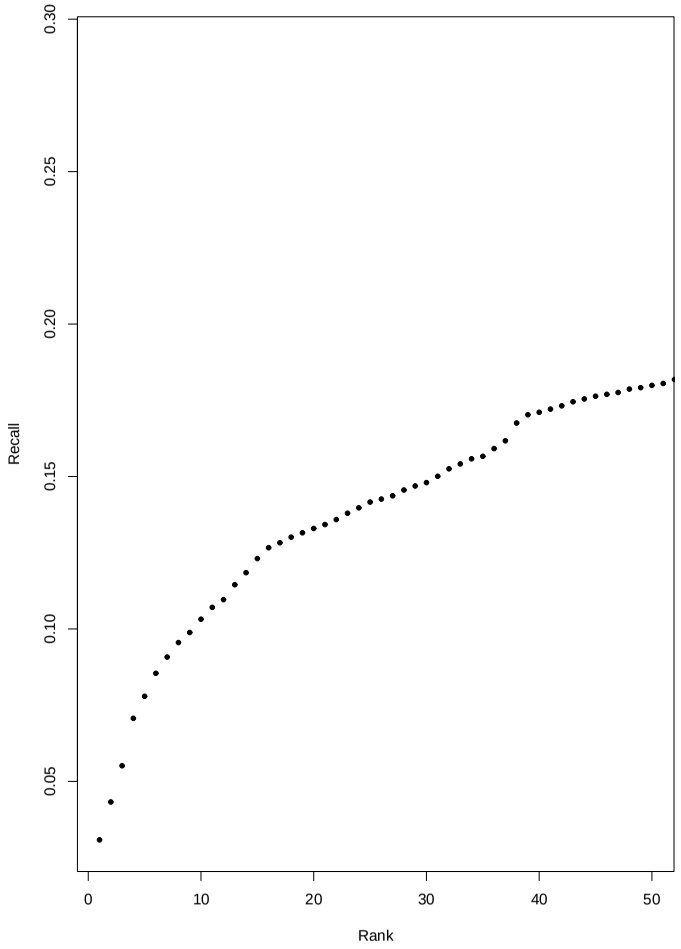

図 12: 50 位以上の候補に対する再現率

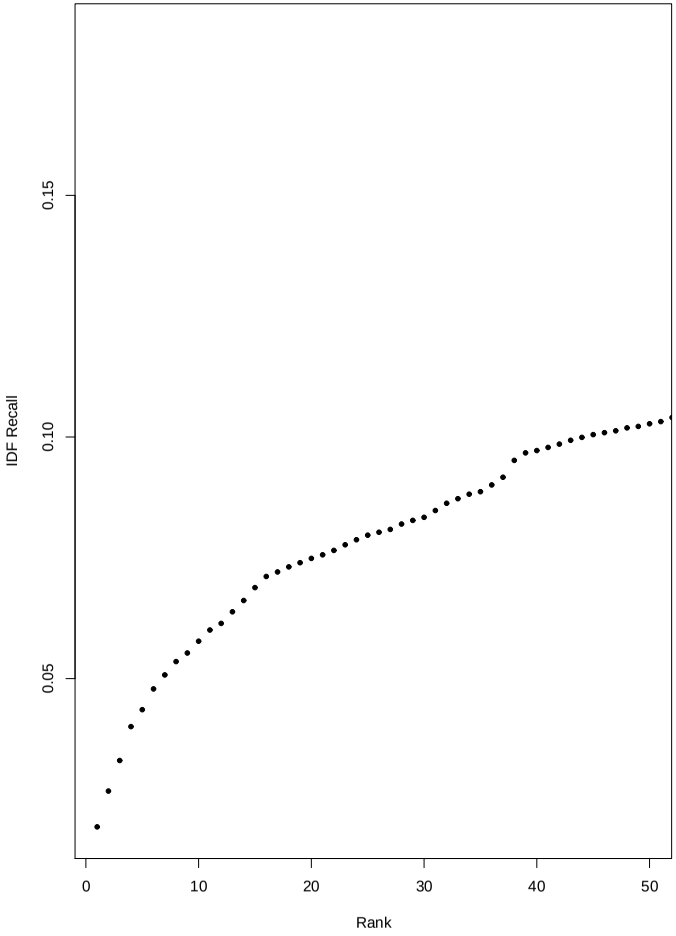

図 13: 50 位以上の候補に対する IDF を考慮した再現率

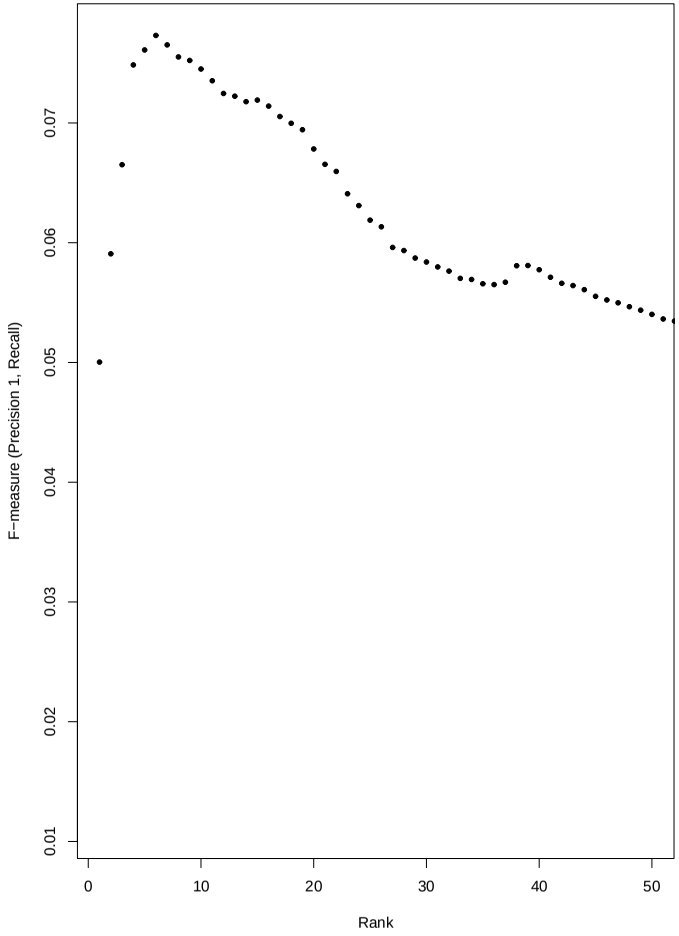

図 14: 50 位以上の適合率 1 と再現率で計算した F 値

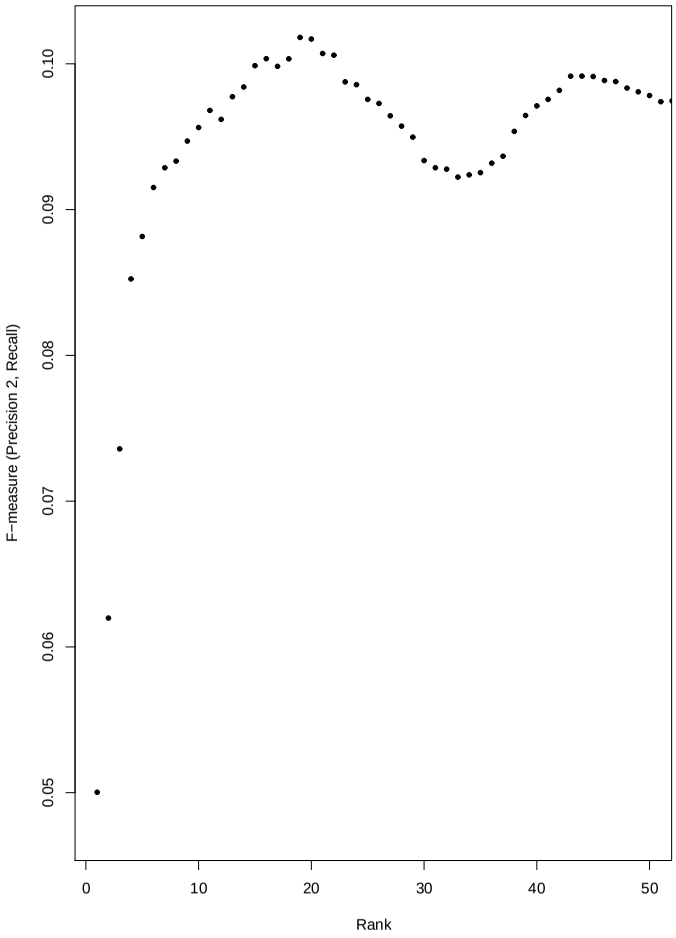

図 15: 50 位以上の適合率 2 と再現率で計算した F 値

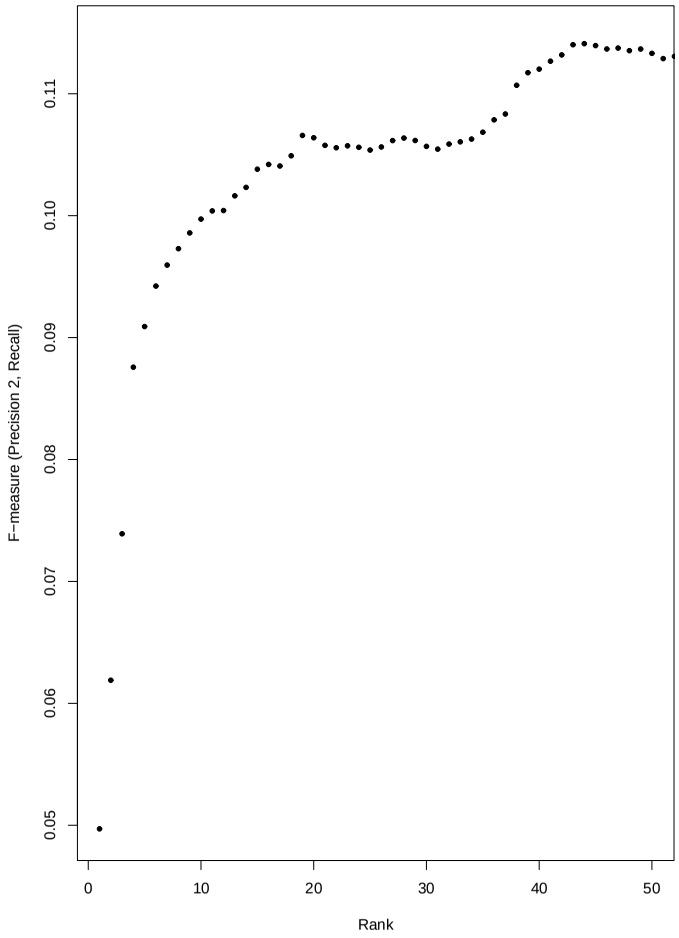

図 16: 50 位以上の適合率 3 と再現率で計算した F 値

表 6: 一部の順位と候補全体における正確な適合率・再現率

	適合率 1	適合率 2	適合率 3	再現率	再現率 (IDF)
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 7: 一部の順位と候補全体における正確な適合率・再現率

	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

6.2.2 RQ2： 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 API 集合を推薦できているか

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 8 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．

表 8: 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

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%

6.2.3 RQ3： 直前に書かれたメソッド呼び出し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連携できるか

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という結果になった．

6.3 考察

本手法は API の推薦を行う既存研究よりも少ない情報から API 集合を推薦しており，比較対象となる既存研究が存在しないため，得られた適合率や再現率の値が良いか悪いかを評価することは難しい．しかし，表 8 を見るとメソッドの動詞とそれ以外の識別子で，検索に使用された割合が大きく異なっていることがわかる．これを改善することにより，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ーフェイ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．

一方，6.1.1 節の適合率の結果を見ると，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，適合率の値が現象する傾向が見られる．すなわち，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がわかる．従って，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おり，それを実現するために既存のソースコード集合から広く使われているライブラリやフレームワークの API 利用実績を学習している．そのため，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は全部で 344,250 個あり，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，実際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 6.1.1 節で示した値より高くなると考えられる．

7　関連研究

本手法は，ソースコード記述中のコード片の推薦・補完と，キーワードに基づく API の推薦の 2 つの側面を持つ．本章ではコード補完を行う手法と，API の推薦を行う手法の 2 つに大きく分けて関連研究を挙げる．

7.1　コード補完手法

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 [23]．この研究では，統合開発環境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メソッド呼び出し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．

Hill らは，入力したメソッドと類似した既存のソースコードのメソッドを提示する手法を提案している [13]．この手法では，メソッドの行数や複雑度に基づいて類似していると判断したメソッドのうち，入力されたメソッドより行数の大きいメソッドを推薦することで，それを見ながらメソッドを記述したり，メソッドを自動で完成させたりするのが目的である．Hill らの手法は入力したメソッドに類似したメソッドを提示するのに対し，本研究ではメソッド本体を入力していない状態でメソッド本体で記述されそうなメソッドを推測して候補を提示する．

Mandelin らは，型に着目したメソッド補完ツール PROSPECTOR を提案している [17]．このツールでは，入力と出力の型をクエリとして，その型に辿りつくための API 列を提示する手法を実装している．この研究が型に着目して API 列を推薦しているのに対し，本研究では主に識別子の名前に着目して API 集合を推薦する．

7.2　API の推薦手法

Wang らは，API を推薦する手法を実装した APIExample [19] 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 [18] 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 [20] や 島田ら [21] は開発環境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開発環境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．

8　まとめと今後の課題

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 の選択に着目した．そこで，本研究では，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実験を行った．この実験によって，並び替えにより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，まず，相関ルールの検索において記述中のソースコードの情報をバランス良く使用し，適した候補を推薦できるように，考察で述べたクラス名の分割などの改善を手法に対して行う必要がある．また，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も行うことを検討している．更に，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

本研究を修士論文として形にすることができたのは，多くの方々に御指導，御協力，御支援を頂いたおかげです．ここに感謝の意を表し，お名前を記させて頂きます．

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 井上 克郎 教授には，研究科長との兼任で非常にお忙しい中，研究方針から論文執筆，発表まで常に適切な御指導御鞭撻を賜りました．私が本研究を続けられたこと，そして楽しく充実した研究生活を送ることができたのは先生のおかげです．心より深く御礼を申し上げます．

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 松下 誠 准教授には，研究発表を通して，研究における問題点や貴重な御意見をいくつも頂きました．先生から頂いた数多くの質問は，研究を客観的に見つめ直すための貴重な材料となりました．深く御礼を申し上げます．

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 石尾 隆 助教には，研究の方針や進め方について非常に多くの御助言を賜り，本論文の作成において熱心に御指導頂きました．また，論文投稿や学会発表の際にも多くの御指導を頂きました．研究を形にできたのは先生のおかげです．深く御礼を申し上げます．

筑波大学システム情報系 早瀬 康裕 助教には，研究室に配属されたばかりの頃から，常々熱心に御指導頂きました．研究について右も左もわからない私に，研究の進め方から文章の書き方，発表資料の作り方，論文の読み方に至るまで，研究を進める上で必要不可欠な知識を一から教えて頂きました．三年間研究を続けることができたのは先生の御指導によるものと確信しております．また，早瀬 康裕 助教には本研究で用いた FP-Tree による部分集合検索アルゴリズムも考案して頂きました．深く御礼を申し上げます．

日本大学工学部情報工学科 山本 哲男 准教授には，本研究を始めるきっかけとなるアイデアを頂きました．また，手法や実装についても多くの御助言を頂きました．実装が完成したのは先生のおかげです．深く御礼を申し上げます．

筑波大学システム情報工学研究科北川データ工学研究室には，FP-Tree による部分集合検索アルゴリズム考案の際，多くの助言を頂きました．私の作成したツールが短時間で動作するためには，このアルゴリズムが欠かせないものとなっております．深く御礼を申し上げます．

また，井上研究室の先輩である，鹿島 悠 氏には，本研究において多くの御支援を頂きました．研究の方針から，論文の執筆，発表資料の作成まで，親身になって相談していただいたことを深く感謝しております．

最後に，研究ではいくつもの壁にぶつかり，時には辛くて逃げ出したくなることもありましたが，この三年間楽しく充実した研究生活を送ることができたのは，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井上研究室の皆様のおかげです．本当にありがとうございました．

参考文献

[1] Code conventions for the java programming language: 9. naming conventions. http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html.

[2] Codewarrior. http://www.freescale.com/codewarrior.

[3] Eclipse. http://www.eclipse.org/.

[4] Netbeans. https://netbeans.org/.

[5] Opennlp. http://opennlp.apache.org/.

[6] Visual studio. http://www.visualstudio.com/.

[7] Wordnet. http://wordnet.princeton.edu/.

[8] Xcode. https://developer.apple.com/xcode/.

[9] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, pages 487–499, 1994.

[10] Tom Brijs, Gilbert Swinnen, Koen Vanhoof, and Geert Wets. Building an association rules framework to improve product assortment decisions. 8(1):7–23, 2004.

[11] Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In *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)*, pages 213–222, 2009.

[12] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000)*, pages 1–12, 2000.

[13] Rosco Hill and Joe Rideout. Automatic method completion. In *Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE 2004)*, pages 228–235, 2004.

[14] Einar W. Høst and Bjarte M. Østvold. Debugging method names. In *Proceedings of the 23rd European Conference on Object-Oriented Programming (ECOOP 2009)*, pages 294–317, 2009.

[15] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Transactions on Software Engineering*, 31(3):213–225, 2005.

[16] Asad Masood Khattak, Adil Mehmood Khan, Tahir Rasheed, Sungyoung Lee, and Young-Koo Lee. Comparative analysis of xlminer and weka for association rule mining and clustering. In *FGIT-DTA*, Communications in Computer and Information Science, pages 82–89, 2009.

[17] David Mandelin, Lin Xu, Rastislav Bodk, and Doug Kimelman. Jungloid mining: helping to navigate the api jungle. In *PLDI*, pages 48–61. ACM, 2005.

[18] Martin Monperrus, Marcel Bruch, and Mira Mezini. Detecting missing method calls in object-oriented software. In *Proceedings of the 24th European Conference on Object-Oriented Programming*, pages 2–25, 2010.

[19] Lijie Wang, Lu Fang, Leye Wang, Ge Li, Bing Xie, and Fuqing Yang. APIExample: An effective web search based usage example recommendation system for java apis. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 592–595, 2011.

[20] Yunwen Ye, Gerhard Fischer, and Brent Reeves. Integrating active information delivery and reuse repository systems. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-first Century Applications*, SIGSOFT '00/FSE-8, pages 60–68, 2000.

[21] 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. 50(12):3095–3107, dec 2009.

[22] 柏原 由紀, 鬼塚 勇弥, 石尾 隆, 早瀬康裕, 山本 哲男, 井上 克郎. 相関ルールマイニングを用いたメソッドの命名方法の分析. ソフトウェア工学の基礎 xx 日本ソフトウェア科学会 FOSE2013. 2013.

[23] 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. 53(2):644–652, 2 2012.