メソッド呼び出し関係に基づくメソッド名の予測

米内 裕史[1,a)]　早瀬 康裕[2,b)]　北川 博之[3,c)]

概要：プログラムの内容や動作を理解する上で，プログラム中の識別子がその役割や動作を示した名前を持つことは，そのソースコードの可読性の重要な要件であるが，識別子に対して動作の手がかりとなり得る適切な名前を付けることは，そのソフトウェアが対象としているドメインの知識を必要とされる，困難な作業である．この問題を解決するため，識別子に対する命名を支援するような手法が提案されている．プログラム中のメソッドに対し，メソッド名の予測を行うような既存の手法では，メソッド名を構成する単語群のうち，名詞部分の予測に関しては動詞部分の予測と比較して精度が低いという課題があった．本稿では，既存手法からの精度の改善，特にメソッド名における名詞部分の精度を改善することを目指し，メソッドの内部の情報からメソッド名として適切な名前を予測する手法を提案する．予測には，メソッドとその内部で使用されているメソッド集合の呼び出し関係を利用する．さらに，抽出した呼び出し関係に基づき，メソッド名に対する分散表現を生成する．メソッド名の予測は，予測対象のメソッドの内部で呼び出されているメソッド集合と獲得した分散表現の位置関係によって候補を決定する．提案手法の有効性を評価するため，既存のソースコード中に出現する各メソッド名に関する分散表現を獲得し予測を行い，実際のメソッド名を提示できるかどうかを検証した．その結果，一定の割合で実際のメソッド名やそのヒントとなるようなメソッド名を提示することができた．

1. 序論

ソフトウェア開発において，開発対象のプログラムを理解する作業は重要である [1]．以前主流であった，開発工程全体を作業工程で分割しトップダウンに進める開発手法では，全作業工程のうちリリース後のソフトウェア保守にかかるコストが約 7 割を占める [2] と言われ，保守対象となるプログラムの修正箇所をソースコードから読み解く必要があり，この作業にかかるコストは保守作業全体のうち大部分を占める [3] と言われている．近年主流となりつつある [4] 開発対象を多数の小さな機能に分割し段階的に変更を加えながらリリースを繰り返す開発手法では，円滑にリリースを続けるためには各機能がどのようなプログラムによって実装されているのかを理解することが重要であるため，プログラムを理解する作業が占めるウェイトはますます大きくなっていると言える．

プログラムを理解する際に，保守作業者はプログラムのソースコード中に出現するクラス名，メソッド名，変数名といった識別子名を動作を理解する手がかりとして利用する [5] ため，開発者はプログラム中の要素に対してその動作や役割を示すような識別子名を与えることで，保守作業者がプログラムを理解するのに要するコストを減らすことができる．しかし，識別子に対してそのような適切に示すような識別子名を与えることは，そのプロジェクトに対する包括的な理解や，識別子の命名に関する知識を必要とする難易度の高い作業である．

以上の背景から，開発者が識別子を命名するタスクを支援するため，ソースコードの情報から識別子名を評価，推薦する手法が提案されている．柏原ら [6], [7] の手法は，メソッドの定義の情報をメソッド名の推薦に利用できることを示したものであるが，メソッド名を構成する単語群のうち，名詞部分の予測に関しては動詞部分の予測と比較して精度が低いという課題がある．Allamanis ら [8] の手法は，この手法はメソッド名やその構成単語に関する分散表現がメソッド名の推薦に利用できることを示したものであるが，分散表現を生成するためにプログラム中でメソッドが呼び出されている文脈を必要とするため，メソッドを定義した時点では名前の推薦を行うことができないという問題

[1] 筑波大学システム情報工学研究科
Graduate school of System and Information Engineering, University of Tsukuba
[2] 筑波大学システム情報系
Faculty of Engineering, Information and Systems, University of Tsukuba
[3] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[a)] yonai@kde.cs.tsukuba.ac.jp
[b)] hayase@cs.tsukuba.ac.jp
[c)] kitagawa@cs.tsukuba.ac.jp

がある．

本研究では，メソッドを定義した時点で可能なメソッド名の予測，および既存手法よりその予測精度，特に名詞部分に関する予測精度の向上を目的として，メソッドの予測を行う手法を新たに提案する．予測の根拠として，メソッドの名前と内部で呼び出しているメソッド集合との間に関連があると考え，既存のソースコードよりそのような関係を抽出し利用する．また，抽出した関係を利用して各メソッド名に対してメソッド名の意味関係の情報を保持するような分散表現を生成し，この位置関係を利用してメソッド名の予測を行う．

提案手法を用いて，既存のソースコードで宣言されているメソッド名に関する分散表現を生成し，これを利用してメソッド名の予測を行い，実際のメソッド名が上位何件目に位置するかを調査した．その結果は，メソッドの呼び出し関係に基いて生成した分散表現が，メソッド名の予測に活用できることを示唆するものであった．また，提案手法が名詞部分に関しても動詞部分と同様の精度で予測できることを示唆するものであった．

本稿では 2 節で本研究における基本事項について説明する．次に 3 節で提案手法について説明する．4 節で提案手法の実装について述べる．5 節で提案手法に関する評価実験について述べる．6 節で実験の結果の考察について述べる．最後に 7 節で本研究についてまとめ，今後の方針について述べる．

2. 基本事項

本節では本研究の基本事項について述べる．2.1 小節では提案手法や一部の既存手法で利用されている単語の分散表現について述べる．2.2 小節ではメソッド名の推薦を目的とした既存手法について述べる．

2.1 単語の分散表現

本節では，本研究や一部の先行研究において識別子名の評価に利用した単語の分散表現について説明する．単語の分散表現は，自然言語の単語を一定数の次元の実数値のベクトルで表す手法 [9] であり，これによって単語を豊富な情報量を保持した上で定量的に表現することができる．例えば，ベクトルの位置関係によって，単語間の意味の類似関係を表現することができる．単語の分散表現は，自然言語処理の分野において活発に利用されており，自然言語における単語に関する分散表現を得る手法が提案されている．Mikolov ら [10], [11], [12] は分布仮説 [13] に基づき，文章中に出現する各単語の前後で出現する単語によって，注目している単語の分散表現を得る手法を提案した．また，この手法によって獲得した分散表現は，$vector(\text{"}King\text{"}) - vector(\text{"}Man\text{"}) + Vector(\text{"}Woman\text{"})$ を計算した結果のベクトルが $vector(\text{"}Queen\text{"})$ と近いものになるなど，ベクトルの位置関係によって意味の関連が表現できたとしている．

2.2 既存手法

開発者が識別子を命名するタスクを支援するため，ソースコードの情報から識別子名を評価，推薦する手法が提案されている．Høst らは，メソッド名を構成する単語のうち動詞部分に注目し，メソッドの処理内容とメソッド名の動詞の関係を調査した．また，そのような関係を収録した辞書を作成する手法を提案した [14]．さらに，メソッド名とメソッドの処理内容の関係を抽出し，抽出したルールに沿わないような不適切なメソッド命名を判別し，より適切なメソッド名を提案する手法を提案した [15]．柏原ら [6], [7] は，メソッドの名前とその内部で使用している変数やメソッド，引数の型などの間に関連があると考え，その間の相関ルール [16] を抽出し，既に定義が存在するようなメソッドに対して，より良いメソッド名を推薦する手法を提案した．この手法はメソッドの定義の情報をメソッド名の推薦に利用できることを示したものであるが，メソッド名を構成する単語群のうち，名詞部分の予測に関しては動詞部分の予測と比較して精度が低いという課題がある．Allamanis ら [8] は，既存のソースコード群からメソッドが呼びされている箇所の前後に出現するプログラム要素から各メソッド名に関する分散表現を生成し，それを利用してメソッド名の予測を行う手法を提案した．また，識別子名を構成要素である単語に分割し，単語それぞれに関して分散表現を生成し，メソッド名を単語の組み合わせと見なして予測を行うことで，既存のソースコード群に含まれていないような新たなメソッド名を提案する手法を提案した．この手法はメソッド名やその構成単語に関する分散表現がメソッド名の推薦に利用できることを示したものであるが，分散表現を生成するためにプログラム中でメソッドが呼び出されている文脈を必要とするため，メソッドを定義した時点では名前の推薦を行うことができないという問題がある．

3. 提案手法

本節では提案手法について述べる．既存の手法における名前における名詞部分の予測精度が低いという課題，メソッドを定義した直後では予測ができないという課題を克服するため，提案手法ではメソッドの定義内の情報を用いてメソッド名に関する分散表現を獲得する．図 1 に提案手法全体の概要図を示す．提案手法は大きく前処理と予測処理の部分に分けられる．前処理では，既存のソースコードよりメソッド定義を抽出しメソッド間の呼び出し関係に基いてメソッド名に対応する分散表現を獲得する．予測処理では，定義が存在するがメソッド名が不明であるようなメソッドが与えられた時に，そのメソッドを表す分散表現を生成し，ベクトル空間上でその分散表現の近傍を探索する．

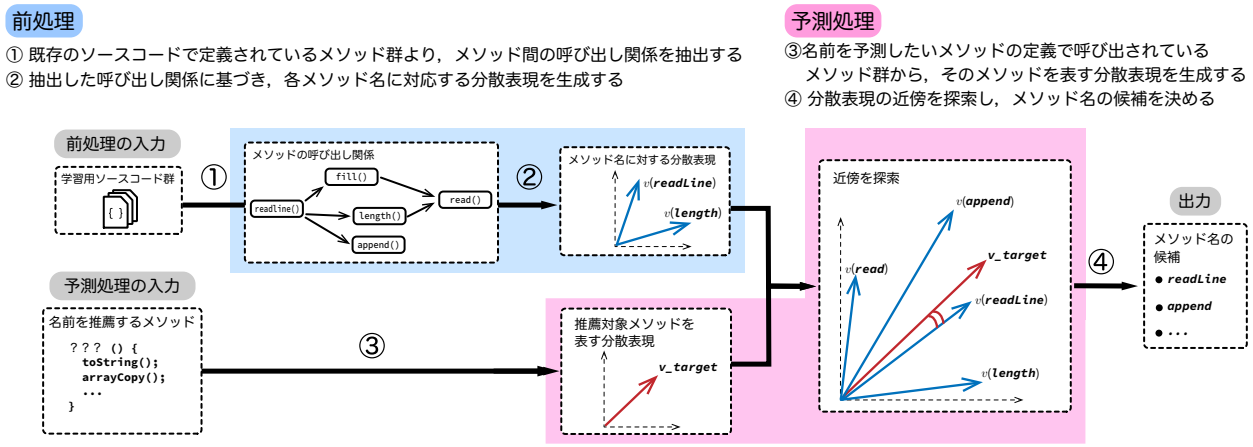

図 1: 提案手法の概要図

この時近傍に存在する既存のメソッド名に関する分散表現をメソッド名の候補とし，予測を行う．

メソッド名の構成要素として使用される名詞は非常に多様であるため，柏原らの手法で利用されていたメソッドの定義の情報とメソッド名の間の相関ルールではそのような多様な名詞から適切な名詞を選択することが難しかった．分散表現を利用することでその位置関係から単語間の意味の関連を表現できるため，予測精度の向上に寄与することが期待される．また，メソッドの使用されている文脈に依存せず，定義の内部の情報のみを用いて分散表現を生成するため，メソッドを定義した時点でメソッド名の予測ができることが期待される．

なお，柏原らの手法はメソッドの定義の情報として，内部で使用されている変数名やメソッド名，引数や返り値の型といったものを利用していた．提案手法でもこれらの情報を利用し分散表現を生成することが望ましいと考えられるが，まずメソッドの定義の情報から分散表現を生成することがメソッド名の予測において有用であることを確認することが望ましい．今回は単純化のため，ソースコード中の要素の中でもメソッドのみに注目し，その間の関係としてメソッド間の呼び出し関係を利用する．即ち，メソッドの内部で呼び出しているメソッドの集合によって，呼び出し元メソッドの名前に関する分散表現を生成する．

また，Allamanis らの手法もメソッド名に関する分散表現を利用するものであるが，Allamanis らはメソッドが呼び出されている箇所の前後のプログラム要素をコンテキストとして分散表現を生成していたのに対し，提案手法はメソッドに対してそのメソッドのボディのプログラム要素をコンテキストとして分散表現を生成しているという点で異なる．

本手法の新規性は，メソッド名を決定する根拠として内部で呼び出しているメソッド集合を利用していること，またメソッド名に対する分散表現をメソッドの呼び出し関係から生成し予測に利用している点である．

メソッド名に対する分散表現を生成するにあたり，多様なメソッドの命名事例およびそれらの呼び出し関係を必要とする．呼び出し関係を収集するため，既存のソフトウェアのソースコード群より，定義されているメソッドとその内部で呼び出しているメソッド集合の関係を抽出した．分散表現の値をメソッド間の呼び出し関係を表すように最適化するため，呼び出し元のメソッド名に対する分散表現の値と呼び出し先メソッドの分散表現の値の関係に関する損失関数を定義し，確率的勾配降下法を用いてこの損失関数を最小化するように分散表現の値を最適化する．

本節では 3.1 小節で提案手法のうち，前処理である分散表現の獲得する部分の手順の詳細について述べる．3.2 小節ではソースコード収集における課題とその解決策について述べる．3.3 小節で予測処理である獲得した分散表現に基いてメソッド名の予測を行う部分の詳細について述べる．

3.1 分散表現の獲得手順

本節では既存のソースコード集合からメソッド名に関する分散表現を獲得する流れについて説明する．

既存のソースコードより，定義されているメソッドとその定義の内部で呼び出されているメソッドの集合を抽出する．抽出にはオープンソースソフトウェアの Java の解析用のライブラリである JavaParser[17] を使用した．この時，各メソッドに関する呼び出し関係を正確に抽出するためには，各メソッドが定義の内部で呼び出しているメソッドがどのパッケージ，クラスに属するものであるのか，また各メソッドはどのパッケージ，クラスに属するメソッドの内部で呼び出されているのかといった，メソッドの名前解決を正確に行う必要がある．ここではそのような呼び出し関係をコールグラフを用いてモデル化する．生成するコールグラフは，各メソッドがノード，呼び出し元のメソッドのノードから呼び出し先のメソッドのノードに有向エッジ

が張られるようなグラフである．このようなコールグラフを，既存のソースコードから抽出したメソッドの呼び出し関係より生成する．

生成したコールグラフの各ノードが示すメソッド名に対応する分散表現の値が，そのノードを始点とするエッジの先のノードが示すメソッド集合，即ち定義の内部で呼び出しているメソッド集合の分散表現の平均の値となるように，各メソッドに対応する分散表現の値を定める．これにより，メソッドの内部の動作，即ち呼び出しているメソッドによって呼び出し元のメソッドの分散表現を決定することを表現する．図 2 に概要図を示す．また，内部で呼び出しているメソッド群の分散表現の平均を取ることにより，呼び出し元のメソッドの分散表現と内部で呼び出しているメソッドの分散表現の間で意味的な関連を表現できることが期待される．

3.2 ソースコード収集における課題と解決策

本手法によって実際にメソッド名の予測を行う場合，予測結果の候補となるメソッド名は分散表現を生成したメソッド群から選出される．そのため多様なメソッド名を提示できるようにするには，その分多様なメソッド定義とメソッド名の命名事例を収集し，分散表現を生成する必要がある．

しかし，大量のソースコードを収集するという状況において，コールグラフの構築にあたり実際にはソースコード中のメソッドの名前解決ができない場合があり，実際には完全なコールグラフを生成するのは困難である．なぜならば，メソッドはクラスやパッケージが異なれば 1 つのメソッド名で複数の定義が存在し得るが，名前解決ができない場合，メソッドのボディで呼び出している他のメソッドがそのメソッド名に対応する複数の定義のうち，どの定義を指しているものなのか区別できないからである．対策として，同名のメソッド定義が複数あった場合に，コールグラフ上でそれらをクラスやパッケージにより区別せずメソッド名のみで識別し，ひとつのノードに統合するという処理を行なう．このとき，統合されるそれぞれのメソッドの定義の内部で呼び出しているメソッド群はすべて統合後のひとつのメソッドが呼び出しているものとする．

3.3 分散表現に基づいたメソッド名予測

メソッド名の予測は，既に定義が存在するがメソッド名が不明もしくは改名するような場面を想定して行う．まずメソッドの定義の内部で呼び出しているメソッド集合を抽出する．抽出したメソッド集合に関する分散表現の平均の値 $v\_avg\_callees$ を計算する．この $v\_avg\_callees$ が呼び出し元のメソッド名に期待する分散表現の値であるので，ベクトル空間上で $v\_avg\_callees$ に近い分散表現をもつメソッド名を呼び出し元メソッドのメソッド名の候補とする．近さの判定は $v\_avg\_callees$ と各分散表現とのコサイン類似度を計算することによって行い，コサイン類似度が高い順に確信度の高い候補とする．

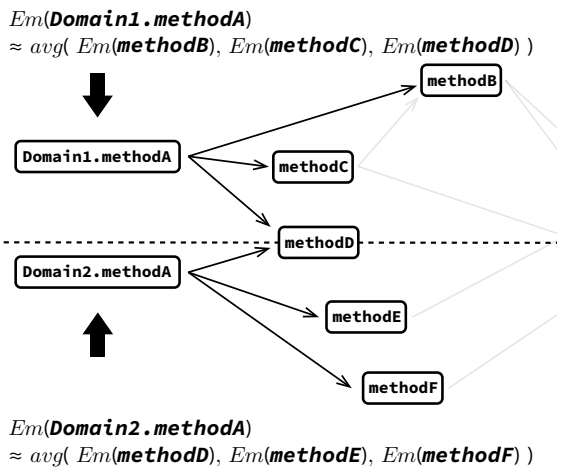

図 2: コールグラフより分散表現の値を決定する例

4. 実装

本節では 3 節で述べた提案手法を実装について述べる．まず 4.1 節で実装の基本的な方針について述べる．4.2 節では分散表現を最適化する際の課題とその解決策を示し，それを反映した実装について述べる．

4.1 実装の基本方針

本小節では，前節までに述べた分散表現の獲得方法に則り，実際に各メソッド名に関する分散表現を獲得する部分の実装について説明する．

まず，獲得する分散表現に対応する関数 E_1 を (1) の通り定義した．

$$E_1 = \sum_{k=1}^{n} |Em(m_k) - avg(Em(t_{k1}) + Em(t_{k2}) + \dots)|^2 \quad (1)$$

関数 E_1 において，n はコールグラフ上のメソッドの数，$Em(m_k)$ は k 番目のメソッドに関する分散表現，$Em(t_{ki})$ は k 番目のメソッドの定義の内部において i 番目に呼び出されているメソッドの分散表現を示す．この関数は各メソッド名に対応する分散表現が，内部で呼び出しているメソッド群の分散表現の平均に近いほど値が小さくなるので，期待する分散表現の獲得は，この関数を最小化するような分散表現の値を求める最適化問題として扱うことができる．

最適化のアルゴリズムには確率的勾配降下法を使用する．今回の問題では，最適化中の各ステップでコールグラフ上のメソッドをランダムにいくつか選択し，そのメソッドに関する損失関数 E_1 の勾配を算出し，分散表現の値の更新を行なう．

ここで，i 番目のメソッド m_i に関する関数 E_1 の勾配は，E_1 を m_i で偏微分することにより，

$$\frac{\partial}{\partial m_i}E_1 = 2Em(m_i) - \frac{2}{|t_i|}\sum_{k=1}^{|t_i|} Em(t_{ij}) \tag{2}$$

となる．このとき，$|t_i|$ は i 番目のメソッドのボディで呼び出されているメソッドの数である．これは，i 番目のメソッドの分散表現が，そのボディで呼び出しているメソッド群の分散表現の平均に近づくような勾配を示している．

また，i 番目のメソッドのボディで j 番目に呼び出されているメソッド t_{ij} に関する関数 E_1 の勾配は，E_1 を t_{ij} で偏微分することにより，

$$\frac{\partial}{\partial t_{ij}}E_1 = -\frac{2}{|t_i|}Em(m_i) + \frac{2}{|t_i|^2}\sum_{k=1}^{|t_i|} Em(t_{ik}) \tag{3}$$

となる．これは，i 番目のメソッドの定義の内部で呼び出しているメソッド群の分散表現の平均が，呼び出し元である i 番目のメソッドの分散表現に近づくような，ボディで j 番目に呼び出されているメソッドに関する勾配である．(2) と (3) は注目してる Caller-Callees の組は同じであるが，(2) は呼び出し元のメソッドに関する勾配であるのに対し，(3) の勾配は呼び出し先のメソッド群のうち 1 つのメソッドに関する勾配であるという違いがある．

以上より，あるメソッドに関する関数 E_1 の勾配の値は，そのメソッドにおける (2) に基づく項と，そのメソッドをボディで呼び出しているいくつかの他のメソッドにおける (3) に基づく項の和である．確率的勾配降下法の各ステップでコールグラフ上のメソッドをランダムにいくつか選択し，選択したメソッドに対して上記の流れで損失関数 E_1 の勾配を計算し，損失関数の値を小さくするように選択したメソッドの分散表現の値を調整するのが実装の基本的な方針である．

4.2 最適化における課題と解決策

前小節で述べたアルゴリズムで分散表現の最適化を行なうにあたり，収束した値によってはメソッド名に関する分散表現として不適切である場合がある．次にその事例を示す．1. 全てのメソッド名に関する分散表現が零ベクトルに向かって収束した場合．関数 E_1 の値は小さくなるが，零ベクトルではメソッド名の役割や動作を表現できたとは言えない．2. 全てのメソッド名に関する分散表現が零ベクトルではなくとも，ベクトル空間上の一箇所に集まるように収束した場合．これも関数 E_1 の値は小さくなり，分布の傾向によっては「分散表現が近いメソッド間は意味の関連が強い」ことは表現できている場合もあるが，「分散表現が離れているメソッド間では意味の関連が弱い」ことを表現できない．

まず事例 1 に対応するため，損失関数 E_2 を (4) の通り定義する．

$$E_2 = \sum_{k=1}^{n} |1 - Em(m_k)|^2 \tag{4}$$

E_2 は各メソッドのノルムが 1 に近づくほど値が小さくなるような損失関数である．この E_2 の値を小さくするように分散表現を最適化することで事例 1 に対応する．これにより，本手法で用いる最終的な損失関数 E は

$$\begin{aligned} E &= \alpha E_1 + (1-\alpha)E_2 \\ &= \alpha \sum_{k=1}^{n} |Em(m_k) - avg(Em(t_{k1}) + Em(t_{k2}) + \dots)|^2 \\ &+ (1-\alpha)\sum_{k=1}^{n} |1 - Em(m_k)|^2 \end{aligned} \tag{5}$$

となる．また，損失関数 E_2 の i 番目のメソッドに関する勾配は

$$\frac{\partial}{\partial m_i}E_2 = 2\frac{|Em(m_i)| - 1}{|Em(m_i)|}Em(m_i) \tag{6}$$

となる．

次に事例 2 に対応するため，本手法における確率的勾配降下法において各ステップで，注目しているメソッドと呼び出し関係の無いメソッドをいくつか選択し，それらのメソッドの分散表現から離れるような勾配の項を追加する．i 番目のメソッドに注目しているとき，呼び出し関係の無いメソッドとして k 番目のメソッドを選択したとすると，k 番目のメソッドの分散表現から離れるような勾配の項 g_{ik} は

$$g_{ik} = -2(o_{Em(m_k)} - Em(m_i)) \tag{7}$$

で表される．ここで，$o_{Em(m_k)}$ は，$Em(m_i)$ と $Em(m_k)$ によって定められる超平面における，$Em(m_k)$ の 2 つの直交ベクトルのうち，$Em(m_i)$ に近いほうのベクトルを指すものである．つまり，この項は i 番目のメソッドに関する分散表現をそのような直交ベクトルに近づけることで，k 番目のメソッドの分散表現から離れるような項である．

これより，i 番目のメソッドに関して呼び出し関係の無いメソッドをいくつか選択したとき，選択したメソッドのインデックス集合を $\boldsymbol{N}$ とすると，呼び出し関係の無いメソッドの分散表現から離れるような項 g_i は

$$g_i = \sum_{\boldsymbol{N}\in k} -2(o_{Em(m_k)} - Em(m_i)) \tag{8}$$

で表される．

以上の 2 つの損失関数の勾配の項を含めた，最終的な分散表現の最適化アルゴリズムを Algorithm1 に示す．

5. 調査実験

本節では提案手法によるメソッド名の予測を評価するために行った実験について述べる．本実験の目的は，提案手

Algorithm 1 ベクトルのノルムに関する項と呼び出し関係の無いメソッドから離れる項を加えた分散表現の最適化

条件:
E は損失関数
w_1，w_2，w_3, w_4 はそれぞれ勾配の項の重みを決めるパラメータ
各メソッドの分散表現をランダムに初期化
loop ループ回数
    $\lambda \leftarrow$ ループ回数に対応する学習率
    $methods \leftarrow$ ランダムにいくつかのメソッドを選択
    for m in $methods$ **do**
        $grad\_callees \leftarrow$(2) に基づく，m の定義内で呼び出しているメソッド集合に関する勾配

        $grad\_callers \leftarrow \mathbf{0}$
        for m を定義内で呼び出しているメソッド集合 **do**
            $grad\_callers \mathrel{+}=$ (3) に基づく，m を定義内で呼び出しているメソッド集合に関する勾配
        end for

        $grad\_norm \leftarrow$(6) に基づく，Em(m) のノルムを 1 に近づけるベクトル

        $grad\_neg \leftarrow \mathbf{0}$
        $neg\_methods \leftarrow$ m と呼び出し関係の無いメソッドをランダムにいくつか選択
        for nm in $neg\_methods$ **do**
            $nm\_ovec \leftarrow$ Em(nm) の直交ベクトル
            $grad\_neg \mathrel{+}=$ (8) に基づく，Em(m) を nm_ovec に近づけるベクトル
        end for
        $Em(m) \mathrel{-}= \lambda(w_1 grad\_callees + w_2 grad\_callers + w_3 grad\_norm + w_4 grad\_neg)$
    end for
end loop

法により生成した分散表現が，メソッド名の予測を行う上で有効であるかどうか，また提案手法によるメソッド名の予測が，メソッドの動詞部分だけでなく，名詞部分に関しても行える可能性があるかどうかを評価することである．この目的を達成するため，提案手法によるメソッド名の予測結果の上位のメソッド名と，予測対象メソッドの実際の名前を比較し傾向を調査する．

実験では，まず既存のソースコード集合を入力としてメソッド間の呼び出し関係に基づき分散表現を生成する．その後，分散表現の生成に利用したソースコード集合で定義されているメソッド集合に対して，3.3 節で述べた手順でメソッド名の予測を行う．各メソッド名に関して，提案手法の出力であるメソッド名の予測結果の中に，実際のメソッド名が予測結果の上位何件目に位置しているかを調査する．また，予測結果の上位 10 件のメソッド名に関して，実際のメソッド名と構成単語が共通するものが存在するかどうかを調査する．なお，メソッド名に関する単語分割は，メソッド名がキャメルケースとスネークケースに基いて構成されているものとして行い，単語の形態素解析には英文の形態素解析ソフトウェアである TreeTagger[18] を使用し，形容詞と判断された単語も名詞として扱うものとする．また，内部でメソッド呼び出しを行わないようなメソッド（コールグラフ上で自身が始点となるようなエッジが存在しないノード）に関しては予測ができないため対象外とする．

上記の調査に加えて，予測に成功するメソッド名と失敗するメソッド名にはどのような違いがあるのかを考察するため，各メソッドが内部で呼び出しているメソッドの数と予測の成否の相関を調査する．本手法ではメソッド名を表す分散表現を，そのメソッドの callee の分散表現の平均に近づくように定義しているため，callee の数が多すぎる場合，各 callee の分散表現のもつ特徴が薄められてしまい，caller のメソッド名の分散表現にその特徴が十分に反映されず期待するような caller の分散表現が得られない可能性があり，予測精度に影響を及ぼすことが考えられる．

また，今回の実験は，予測対象のメソッドの名前と予測結果の上位 10 件のメソッド名のいずれかに 1 単語でも共通するものがあれば良い，という判定基準で行うため，予測対象のメソッドの名前を構成する単語の数が多い場合，精度評価に有利に働く可能性がある．そこで，各メソッド名を構成する単語の数と予測の成否の相関に関しても調査を行う．

本実験で分散表現の獲得や予測に使用するソースコード群は，分散表現を利用してメソッド名の予測を行うような先行研究である Allamanis ら [8] の手法において評価実験に使用していたデータセットを使用する．このデータセットは [19] において他者が利用できる形で公開されている．このデータセットはソフトウェア開発プロジェクトを共有するサービスである GitHub[20] に公開されている Java を用いたプロジェクト群を Allamanis らが定めた，リポジトリ

の Watcher 数と Fork 数に基づく指標によって選定し，ソースコードを取得したものである．このデータセットに含まれるメソッドを 3.2 節で述べた同じメソッド名に関する複数の定義を統合する処理を行った．その後にデータセットに含まれていたメソッドの数は 84,408 個，そのうち予測対象となるメソッドの数は 61,259 個である．

5.1 実験結果

実験の結果，実際のメソッド名が予測結果の上位何件目に位置するかに関する割合を図 3 に示す．この図において値 k は予測結果における実際のメソッド名の順位の値を示す．予測対象である全 61259 個のメソッドのうち，24.2 %にあたる 14852 個のメソッドに関して，予測結果の上位 10 件に実際のメソッド名が存在した．

また，予測結果の上位 10 件に関して，実際のメソッド名と動詞部分が一致するメソッド名が存在するかどうか調査した結果を図 4 に示す．予測対象となったメソッドのうち，その 64.8 %にあたるメソッドに関して，予測結果の上位 10 件に実際のメソッド名と共通の動詞部分をもつメソッド名が存在した．また，同様に名詞部分が一致するメソッド名が存在するかどうか調査した結果を図 5 に示す．69.7 %にあたるメソッドに関して，予測結果の上位 10 件に実際のメソッド名と共通の名詞部分をもつメソッド名が存在した．

この結果は柏原ら [6], [7] の手法と比較した場合，柏原らの手法を評価する実験で使用されていたデータセットと本実験で使用したデータセットでは，含まれるリポジトリや含まれるメソッドの定義の数，評価の方法が異なるため厳密な比較は難しいが，柏原らの実験ではメソッド名の動詞部分と名詞部分を上位 100 件に予測できた割合がそれぞれ約 5 割と約 1 割であったため，本手法は優れた精度で予測ができる可能性があると考えられる．

同様に Allamanis ら [8] の手法との比較も行うべきであるが，いくつかの理由で困難である．Allamanis らは論文において手法の評価に *Suggestion Frequency* と呼ばれる指標を導入している．これは Allamanis らの評価実験において，テストケース内のメソッド名の集合に関する，あらかじめ設定された閾値より高い予測の確信度で予測結果を提示できたメソッド名の割合である．Allamanis らの手法はメソッド名の予測を行う際に，その予測した名前の確信度を同時に提示するものであるが，このとき確信度の低い予測を提示することはユーザーを戸惑わせてしまうと考えられるため，確信度の値に閾値を設定し，その値以上の確信度をもつ予測結果のみをユーザーに提示することを想定している．Allamanis らの評価実験はこの *Suggestion Frequency* を値を変化させたときに，予測結果の精度の F 値がどのように変化するかという指標で行っているが，手法の比較が困難である理由の 1 つは，このとき比較している 4 つのモデルのうち 3 つは *Suggestion Frequency* の値が一定以上の場合に予測結果の確信度がすべて閾値以下で，何も提示できず評価が行えなかったという結果であったためである．もうひとつの理由は，Allamanis らの 4 つのモデルのうち，*Suggestion Frequency* の値に関わらず結果を提示できた残りの 1 つのモデルはメソッド名を構成単語に分割し，単語ごとに予測を行っているモデルであるが，このとき F 値をどのように計算しているかに関して具体的に述べられていないためである．

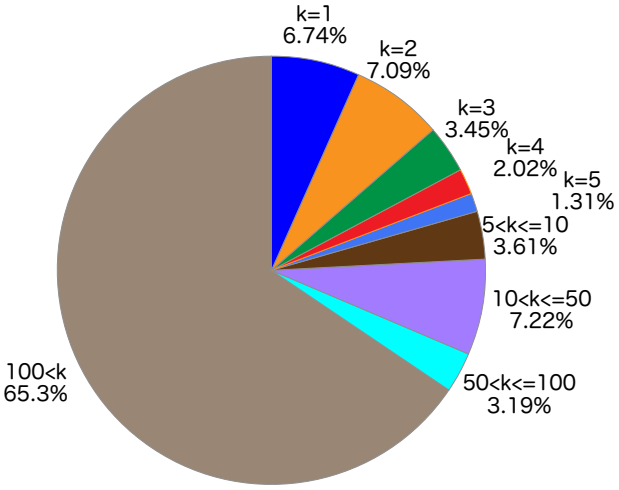

図 3: メソッド名の予測結果における実際のメソッド名の順位 k ごとの割合

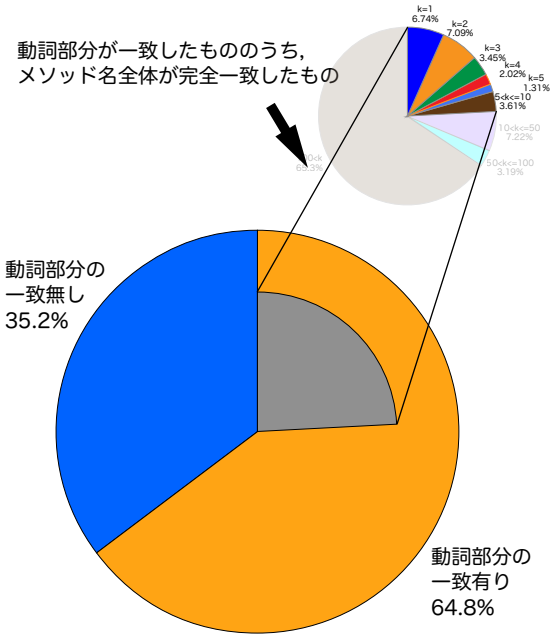

図 4: 予測結果上位 10 件に動詞部分が一致するメソッド名が存在した割合

上記の予測の結果に関して，メソッドの内部で呼び出しているメソッドの数との相関を調査した．図 6 は，メソッドの callee の数のごとの予測結果上位 10 件における動詞

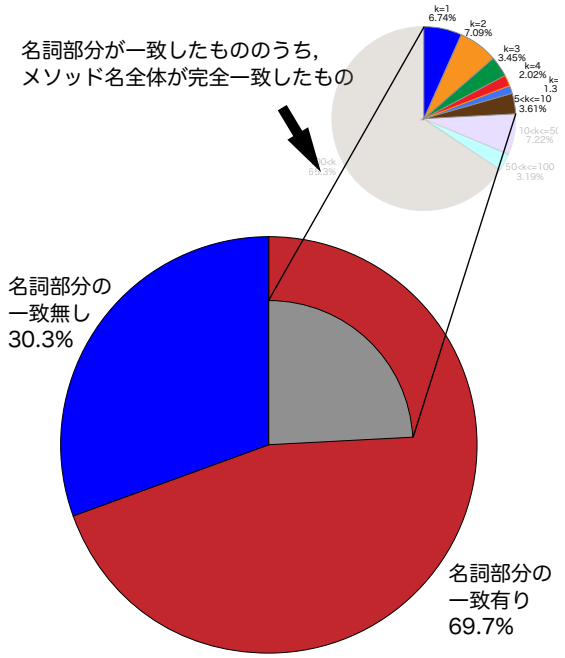

図 5: 予測結果上位 10 件に名詞部分が一致するメソッド名が存在した割合

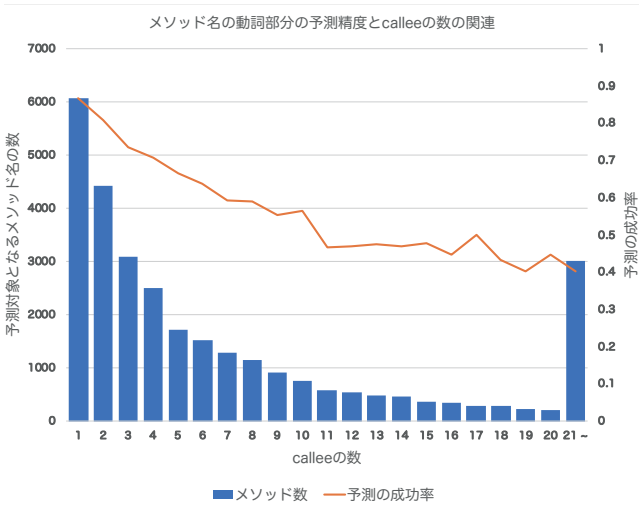

図 6: メソッドの callee の数と予測結果上位 10 件における動詞部分の予測精度及びメソッド数の相関

部分の予測の精度とメソッド名の数の関係を示したものである．この結果から，callee の数が多くなるにつれて，動詞部分の予測精度が低下する傾向があることが分かる．図 7 は同様に名詞部分の精度に関して調査したものであり，こちらも動詞の場合と同様の傾向があると判断できる．

また，同様に予測の結果に関して，メソッド名を構成する単語の数との相関を求めた．図 8 は，元のメソッド名を構成する単語の数ごとの予測結果上位 10 件における動詞部分の予測の精度とメソッド名の数の関係を示したものである．この結果から，元のメソッド名の構成単語数が多くなると，予測精度は上下に大きく変化するため，メソッド名の構成単語数と動詞の予測精度には大きな相関は無いと

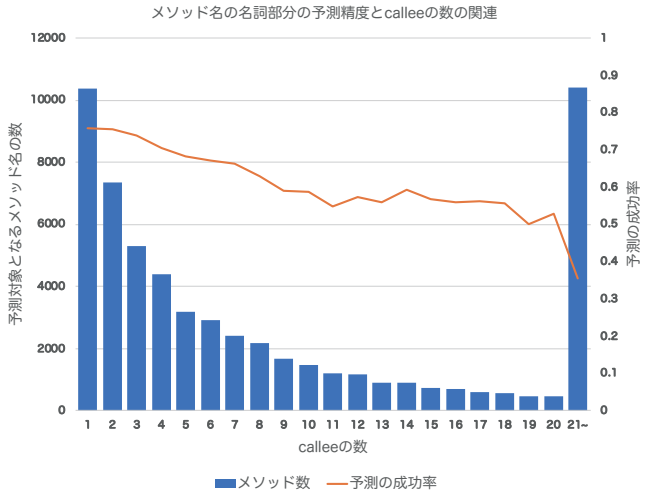

図 7: メソッドの callee の数と予測結果上位 10 件における名詞部分の予測精度及びメソッド数の相関

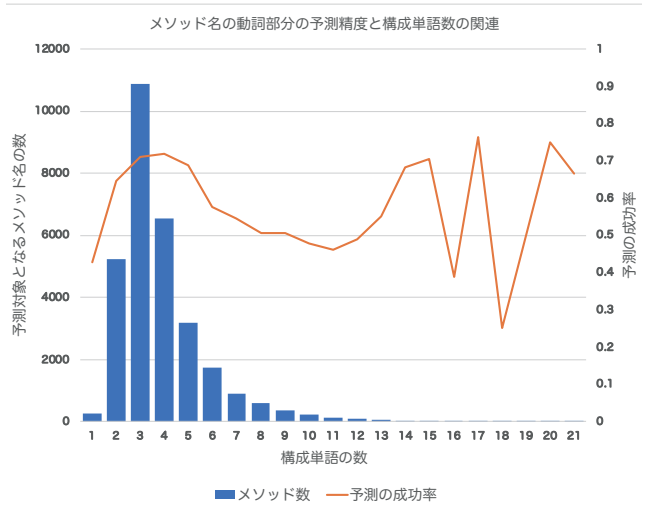

図 8: メソッド名を構成する単語の数と予測結果上位 10 件における動詞部分の予測精度及びメソッド数の相関

考えられる．図 9 は同様に名詞の一致に関して調査したものであり，こちらは構成単語数が多くなるにつれて予測が成功するメソッド名の割合が大きくなるような相関があることが分かる．

6. 考察

全体のメソッド名の数が約 6 万件存在する中で，約 4 分の 1 のメソッド名について上位 10 件に実際のメソッド名が存在したという結果は，メソッドの呼び出し関係に基いて生成した分散表現が，メソッド名の予測に利用する上で有効であることを示唆するものであると言える．しかし，全体の約 4 分の 3 のメソッドにおいて，実際のメソッド名を上位 10 件に含めることができなかった．原因の 1 つとして，メソッドの候補の数が挙げられる．本手法は，データセットに含まれる全てのメソッド名に関して分散表現を生成しているため，メソッド名の予測を行う際，そのすべ

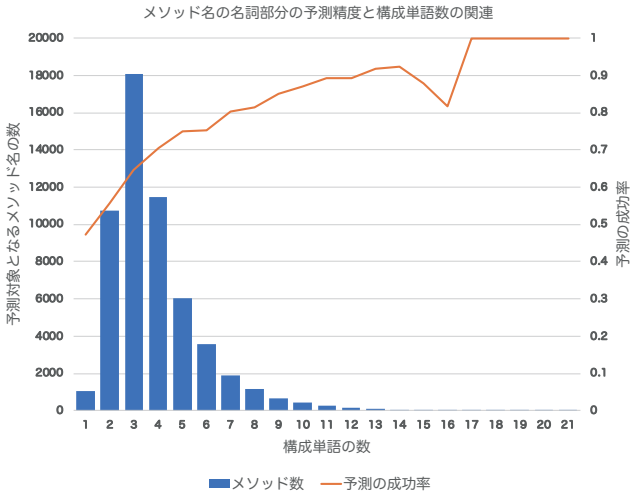

図 9: メソッド名を構成する単語の数と予測結果上位 10 件における名詞部分の予測精度及びメソッド数の相関

てのメソッド名が予測の候補となるが，メソッド名の候補の数が極端に多い場合，良いメソッド名を上位に出すことが難しくなっていると考えられる．分散表現を生成するメソッド名の数を減らすことにより高精度で予測できるメソッドの割合を大きくできる可能性があるが，予測できるメソッドのバリエーションを減らすことになるため部分的な予測の精度も下がる可能性がある．

また，メソッド名の動詞部分，名詞部分の両方において，約 7 割のメソッド名に関して実際のメソッドと共通する単語をもつメソッド名を上位 10 件に予測することができた．先述の実験結果よりこのうち 24.2 %のメソッドは実際のメソッド名を予測できたものであるため，実際のメソッド名ではなくともその名前を構成する単語を部分的に予測できた割合は全体の約 45 %である．この結果は，提案手法がメソッド名の動詞部分と名詞部分の両方に関して予測できる可能性があることを示唆していると言える．なお，今回の評価ではメソッド名を構成する単語の形態素解析に英文の形態素解析ツールを利用した．メソッド名を単語分割することにより得られる単語列は，通常の英文として見なした場合に構文が不適切である場合が存在するため，形態素の解析が期待通りに行われていない場合が存在する可能性がある．また，一般的な英語としては動詞として使われる場合がほぼ存在しない単語が，メソッド名を構成する単語として使用した場合に動詞のような振る舞いで使用される場合がある．例として Java において広く使われるメソッド名である”toString”における”to”が挙げられる．”toString”は何らかのオブジェクトを String 型に変換することを示唆するメソッド名であるが，このメソッド名において”to”は「変換する」という動作を表す動詞のように振る舞っていると言える．よって，通常の英文の形態素解析の手法から，よりメソッド名における構文に則った形態素解析を行う必要があると考えられる．

予測の精度とメソッドの内部で呼び出しているメソッドの数との相関に関しては，内部で呼び出しているメソッドの数が多くなるにつれて，動詞部分・名詞部分共に予測精度が低下する傾向があった．このことから，メソッドの callee の数が，メソッド名の予測の成否に影響を与える可能性があると言える．また，メソッドの callee の集合から，そのメソッドの分散表現に反映させる上で効果的な callee をいくつか選択し，選択した callee の集合の平均によって呼び出し元のメソッドの分散表現を決定することで，予測精度が向上する可能性があると考えられる．

予測の精度とメソッド名を構成する単語の数の相関に関しては，動詞部分の予測精度に関しては有意な相関が無かった．これは，Java の識別子命名における一般的な慣例 [21] においては，メソッド名の構成単語数がいくつの場合であっても，その中に含まれる動詞は多くの場合で 1 つのみもしくは少ない数であるからと考えられる．また，名詞部分の予測精度に関しては構成単語数が多くなるにつれて，精度が向上する傾向があった．名詞の場合はメソッド名の構成単語数が多くなればそれに応じて名詞の数も多くなるため，予測が成功する確率も高くなると考えられる．しかし，そのような構成単語数が多いメソッドはそもそも数が多くないため，この有無が調査結果全体の傾向を大きく変えるものではないと考えられる．

7. 結論

本研究では，メソッド名の命名が難しいという問題に対し，ボディが既に存在するようなメソッドを改名するような場面を対象として，そのメソッドの名前を予測，推薦する手法を提案した．予測に際して，メソッドのメソッド名とそのボディで呼び出されているメソッド集合の関係を利用した．また，収集したメソッドの呼び出し関係に基いて，各メソッド名に関する分散表現を生成し，予測に利用した．

提案手法に利用して，既存のソースコード集合に対して各メソッドのボディで呼び出されているメソッド集合と生成した分散表現から，メソッドの名前を予測する実験を行い，提案手法を評価した．その結果，提案手法によって生成した分散表現によりメソッド名の意味の関連を表現できること，またメソッド名の動詞部分だけでなく名詞部分に関して予測する能力があること示唆する結果が得られた．この結果は提案手法によって一定の割合のメソッドに関して，開発者の命名のヒントとなるような情報を提供できることを示唆している．

今後の方針として，より実際のメソッド名推薦の場面に準じた評価として，分割交差検証などを用いて予測の対象とするメソッド集合を分散表現の生成源となったメソッド集合でないものを使用するような実験を行うことが挙げられる．また，本手法の発展として，今回は単純化のためメソッド定義の内部で呼び出しているメソッドのみに注目し

たが，変数や引数の型といった呼び出しメソッド以外の要素も利用して分散表現の生成に利用することが考えられる．また，より実用的な推薦手法として，メソッド名を構成単語に分割し別々に推薦を行うような推薦手法を検討すること，実際の開発の現場で利用できるような推薦システムを検討することが挙げられる．

参考文献

[1] Von Mayrhauser, A. and Vans, A. M.: Program comprehension during software maintenance and evolution, *Computer*, Vol. 28, No. 8, pp. 44–55 (1995).
[2] Lientz, B. P., Swanson, E. B. and Tompkins, G. E.: Characteristics of application software maintenance, *Communications of the ACM*, Vol. 21, No. 6, pp. 466–471 (1978).
[3] Murphy, G. C., Kersten, M., Robillard, M. P. and Cubranic, D.: The emergent structure of development tasks, *ECOOP*, Vol. 5, Springer, pp. 33–48 (2005).
[4] Larman, C. and Basili, V. R.: Iterative and incremental developments. a brief history, *Computer*, Vol. 36, No. 6, pp. 47–56 (2003).
[5] De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A. and Panichella, S.: Using IR methods for labeling source code artifacts: Is it worthwhile?, *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, IEEE, pp. 193–202 (2012).
[6] 柏原由紀：メソッド名とその周辺の識別子の相関ルールに基づくメソッド名変更支援手法，大阪大学基礎工学部情報科学科特別研究報告 (2013).
[7] Kashiwabara, Y., Onizuka, Y., Ishio, T., Hayase, Y., Yamamoto, T. and Inoue, K.: Recommending verbs for rename method using association rule mining, *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, IEEE, pp. 323–327 (2014).
[8] Allamanis, M., Barr, E. T., Bird, C. and Sutton, C.: Suggesting accurate method and class names, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, pp. 38–49 (2015).
[9] Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C.: A neural probabilistic language model, *Journal of machine learning research*, Vol. 3, No. Feb, pp. 1137–1155 (2003).
[10] Mikolov, T., Yih, W.-t. and Zweig, G.: Linguistic regularities in continuous space word representations., *hlt-Naacl*, Vol. 13, pp. 746–751 (2013).
[11] Mikolov, T., Chen, K., Corrado, G. and Dean, J.: Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781* (2013).
[12] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J.: Distributed representations of words and phrases and their compositionality, *Advances in neural information processing systems*, pp. 3111–3119 (2013).
[13] Harris, Z. S.: Distributional structure, *Word*, Vol. 10, No. 2-3, pp. 146–162 (1954).
[14] Host, E. W. and Ostvold, B. M.: The programmer's lexicon, volume I: The verbs, *Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on*, IEEE, pp. 193–202 (2007).
[15] Høst, E. W. and Østvold, B. M.: Debugging method names, *European Conference on Object-Oriented Programming*, Springer, pp. 294–317 (2009).
[16] Agrawal, R., Imieliński, T. and Swami, A.: Mining association rules between sets of items in large databases, *Acm sigmod record*, Vol. 22, No. 2, ACM, pp. 207–216 (1993).
[17] : JavaParser, `https://javaparser.org/`.
[18] : TreeTagger, `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`.
[19] : Suggesting Accurate Method And Class Names, `http://groups.inf.ed.ac.uk/cup/naturalize/`.
[20] : GitHub, `https://github.com/`.
[21] : The Java® Language Specification, `https://docs.oracle.com/javase/specs/jls/se9/html/`.