Vol.2015-SE-187 No.3
2015/3/12

動詞に着目した相関ルールを利用するメソッド名の命名支援手法の評価

柏原 由紀[1,a)]　石尾 隆[1,b)]　井上 克郎[1,c)]

概要：ソースコードの読解においてソースコード中の識別子がその役割を正しく表現していない場合，プログラムの理解に時間がかかることが知られている．オブジェクト指向プログラムのメソッド名は一般に，動詞や名詞などを組み合わせて命名されるが，具体的にどのようなメソッドに対してどの単語を使って命名したほうがよいかということは知られていない．我々の先行研究として，開発者に対するメソッドの命名支援を目的として，メソッド名に用いる動詞の候補リストを提示する手法を提案した．しかし先行研究では，実際に開発者が候補リストを使用した場合の効果は評価していない．そこで本研究では，動詞の候補リストが提示されたとき開発者がより適切な命名をできるのか評価した．先行研究で言及した課題に基づきメソッド名に用いる動詞の候補リストを提示する既存手法の改良を行い，改良した手法を用いて評価を行った．その結果，既存手法よりも精度が上がったことを確かめ，また，開発者は候補リスト内に正解が提示されているとき候補リストがないときよりも正解の動詞を選択できることを示した．

1. はじめに

ソースコードの読解においてソースコード中の識別子がその役割を正しく表現していない場合，プログラムの理解に時間がかかることが知られている [1]．開発者は，ソースコード片の役割をその内部に出現する識別子から推測することがあり，保守作業の対象を特定する手掛かりの 1 つとしても識別子が重要であると言われている [2]．さらに，識別子の命名においてその役割を過不足なく表す命名を行うことが重要であると複数のガイドライン [3][4] で主張されていることからも，適切な識別子をソースコード中で用いることが重要であることがわかる．

オブジェクト指向プログラムのメソッド名は一般に，その動作と対象を表すように，動詞や名詞などを組み合わせて命名されるが [3]，一般的なガイドラインでは get，set，test といった広く知られている特定の動詞の使い方以外は，メソッドに対してどの単語を使って命名したほうがよいかということは明記されていない．メソッド名に使われる一部の動詞については，メソッドの中に記述されているソースコードの特徴が調査されているが [5]，これ以外の動詞についてはどのようなメソッドに対してどのような語を使うべきか知られていない．

我々の先行研究として，開発者に対するメソッドの命名支援を目的として，メソッド名に用いる動詞の候補リストを提示する手法を提案した [6]．相関ルールマイニング [7] を用いることでメソッド本体とメソッド名の動詞の関係をルールとして抽出し，抽出したルールを対象のメソッドに適用することでメソッド名の動詞の候補リストを提示する手法である．しかし，先行研究では開発者が候補リストを提示されたときの効果は評価していない．

そこで本研究では，メソッド名に用いる動詞の候補リストを提示する既存手法の改良を行い，改良した手法を用いて被験者実験を行った．先行研究で言及していた課題に基づき，ルールの抽出方法について，別の要素を追加することで手法の改良を行った．

本研究ではまず，既存手法と比べてどのくらい改良されたかを評価した．112 個のオープンソースソフトウェアからそれぞれルールを抽出し，同じソフトウェアに対して手法を適用することで，どのくらい手法がされたのかを比較した．その結果，既存手法では，83.1%のメソッドに対して正解の動詞を提示し，39.7%のメソッドについてリストの上位 5 位以内に正解を提示していた一方で，本手法では，86.8%のメソッドに対して正解の動詞を提示し，48.2%のメソッドについてリストの上位 5 位以内に正解を提示できたことがわかった．

また，本手法を用いて，開発者が候補リストを用いたと

[1] 大阪大学大学院情報科学研究科
Osaka University, Suita, Osaka 1–5, Japan
[a)] k-yuki@ist.osaka-u.ac.jp
[b)] ishio@ist.osaka-u.ac.jp
[c)] inoue@ist.osaka-u.ac.jp

きにどのくらい適切な動詞を選択できるのかを評価した．被験者にメソッドを読んでもらい，候補リストがある場合とない場合でどのくらい適切な動詞を選択できるのかという実験を行った．その結果，候補リストがある場合の方が正解数は多かったものの，有意な差は見られなかった．一方で，開発者は候補リストに正解が提示されているとき，候補リストがないときよりも正解の動詞を選択できることがわかった．

本稿における貢献は以下のとおりである．

- 既存手法の改良版を提案した．
- 開発者は候補リストに正解が提示されているとき，候補リストがないときよりも正解の動詞を選択できることを示した．

以降，2 章では関連研究について述べ，3 章では提案手法について説明する．その後，4 章では評価実験について説明し，5 章で妥当性の脅威について議論した後，最後に 6 章でまとめと今後の課題について述べる．

2. 関連研究

先行研究として，開発者に対するメソッドの命名支援を目的として，メソッド名に用いる動詞の候補リストを提示する手法を提案した [6]．メソッド名を動詞と目的語の組とみなし，メソッドを表す情報としてメソッド内に出現する識別子およびその型を利用する．これらに対して相関ルールマイニング [7] を用いて既存のソフトウェア集合からメソッド本体とメソッド名に使われる動詞の関係を相関ルールとして抽出した．抽出したルールを対象のメソッドに適用することでメソッド名の動詞の候補リストを提示する手法である．先行研究では今後の課題として，フィールドの型などの要素を追加すること，相関ルールの評価指標の一つである Lift 値を利用することなどが挙げられている．また，開発者が候補リストを提示されたときの効果については評価していない．

開発者が適切なメソッドを命名できるように支援を行う研究がいくつか行われている．Høst らは，メソッド本体の動作に対してつけられているメソッド名の動詞が非常に不適切であるメソッドを検出した上で適切な動詞を開発者に提示する手法を提案している [8]．また，Høst らの手法は，Kerlsen らによって統合開発環境 Eclipse のプラグインとして実装されている [9]．この手法は，メソッド名のうち特にメソッド名の動詞に着目しており，既存のソフトウェアから作成したメソッド本体と動詞の対応関係を表すルールを用いて，対象のメソッドの動作とメソッド名の動詞がルールをまったく満たさないときに，そのメソッド名の動詞が不適切であると判断する．そのとき開発者に警告すると同時に，メソッド名の動詞の修正候補を提示する．Høst らの手法は，非常に高い精度で不適切なメソッドの検出・修正候補の提示をするものの，適用できるメソッドが非常に限定的であるという問題がある．適用できるメソッドは，事前にルールを作成している動詞のパターンに対してのみで，その数は 76 パターンである．また，メソッドの動作とメソッド名の動詞を一対一で対応付けるルールを用いているため，適用できるメソッドの数を増やしにくいという問題がある．

Yu らは，メソッドの動詞と目的語それぞれに着目し，メソッドの命名を自動で行う手法を提案している [10]．機械学習の一種であるサポートベクターマシン (SVM) の手法を用いて既存のソフトウェアからメソッド本体とメソッド名の動詞の関係を学習し，対象のメソッドに対して適切な動詞を 1 つ選択する．Høst らの手法と異なり，既についているメソッド名を利用していないため，メソッド本体が記述してあれば，どのようなメソッドにも動詞を提示できる．また，Yu らの手法では 237 個の動詞に対応している．ただし，Yu らの手法では，SVM を用いているため，動詞を 1 つしか選択できない．もし開発者が，彼らの手法が選択した動詞はそのメソッドに対して適切でないと判断した場合，開発者はメソッドの命名のために参考にできる情報をそれ以上得ることができない．

3. 手法

本章では，あるメソッドに対して，命名に用いる動詞の候補リストを推薦する手法について説明する．あらかじめメソッド本体とメソッドの命名に用いる動詞の関係を，相関ルールマイニングを用いて抽出しておき，抽出したルールを用いて，対象のメソッドに動詞の候補リストを提示する．候補リストの上位に提示されている動詞ほど，そのメソッドの内容を表す動詞である可能性が高いことを示す．

手法はルールの抽出を行うステップ 1 と，抽出したルールを用いて対象のメソッドに動詞を推薦するステップ 2 から成る．ステップ 1 は，前処理として 1 度だけ行い，その後，入力に与えるメソッドを変更しながらステップ 2 を繰り返し実行する．既存手法に対して，ステップ 1 に変更を加えた．

3.1 ステップ 1: ルールの抽出

Java で記述されたソフトウェアのバイトコード集合を入力として，相関ルールマイニングによって，メソッド本体とメソッド名に用いる動詞の関係をルールとして抽出する．まず，バイトコード集合から，学習に用いるメソッドのみを選択し，メソッド集合を作成する．本手法では，メソッド名の動詞を提示するため，「先頭が小文字である」かつ「先頭の単語として動詞が出現している」メソッドのみを学習に用いた．これは，Java において camelCase で記述されることが命名規約に示されているため，大文字から始まるような名前がついたメソッドは動詞と目的語の組として命名されているかどうかの意図が不明であると考えたた

```
public class NameList implements Serializable {
  String[] nameArray;
  int size;

  public boolean containsName(String n){
    for(int i=0; i<size; i++){
     if(nameArray[i].equals(n)){
       return true;
     }
    }
    return false;
  }

  public void setSize(Integer size) {
    this.size = size;
  }
}
```

(a) ソースコード

メソッド名の動詞 : contains
返り値の型 : boolean
引数の型 : String
フィールドの型 : String[]
フィールドの型 : int
語 : name
語 : array
語 : size
動詞 : equals

(b) containsName メソッドのトランザクション

メソッド名の動詞 : set
返り値の型 : void
引数の型 : Integer
フィールドの型 : int
語 : size

(c) setSize メソッドのトランザクション

図 1 トランザクションの取得例

めである．snake_case で記述されているようなメソッドについては，動詞と目的語の組として命名するという意図は同一であるとし，学習に用いるものとした．また，メソッド名が動詞と目的語の組で命名されることが多いため，先頭の単語として動詞が出現しているメソッドのみを学習に用いた．Yu らも，動詞から始まるメソッド名のみを対象にしている [10].

連結された単語の分解は，キャメルケースもしくはスネークケースで単語が連結されているものとみなして分解を行った．動詞の判定には WordNet[*1] を利用した．ただし，to，new，init，calc，cleanup，setup の 6 つの単語については，メソッドの命名において慣習的に動詞として用いられていることから，品詞解析の結果にかかわらず本手法では動詞として扱う．

また，Java プログラムでは，言語仕様や標準ライブラリの設計によっていくつかのメソッドについては命名の規則が既に確立している．そのようなメソッドとして，以下に示すメソッドを学習セットから除外した．

main メソッドおよびコンストラクタ　言語仕様によって名前が決まっているメソッドであるため．

匿名クラス内に定義されているメソッド　ほとんどが既存ライブラリのオーバーライドであるため．

get，set の動詞が使われているメソッド　フィールドの読み書きを行うメソッドの名前として既に一般的な利用法が確立しているため．

test の動詞が使われているメソッド　JUnit において，テストを表現するメソッドに用いることが確立されているため．

toString，hashCode，equals メソッド　いずれも Java の基底クラスである java.lang.Object に定義されたメソッドであり，オーバーライドによって使用されるメソッドであるため．

本手法ではバイトコード集合から情報を取得している．コンパイル時にデバッグ情報が組み込まれていない場合，メソッド内の引数の名前などの情報が欠落することがある．そのようなメソッドは学習に適していないと考え，そのようなメソッドも学習セットから除外した．

トランザクションはメソッド名の動詞とメソッド本体に出現する単語および型の集合から成り，対応する各メソッドに対して一意に定まる．トランザクションの要素は，そのメソッドに出現したある識別子に使われている単語および識別子の型とその種別の組によって表され，「種別 : 単語」のように記述される．トランザクションに含まれる要素の種別として以下の 6 つを用いた．

メソッド名の動詞　メソッド名に使われている動詞．

返り値の型　メソッドの返り値の型の名前．

引数の型　メソッド引数の型の名前．

フィールドの型　メソッドの中でアクセスしているフィールドの型の名前．

動詞　メソッドの中で呼び出しているメソッドの名前に使われている動詞．「メソッド名の動詞」と同様，先頭に出現する動詞のみを呼び出しメソッド名の動詞として扱う． 先頭が動詞でないものについては，すべての単語を以下で説明する「語」として扱う．

語　引数の名前，フィールドの名前に使われている単語，および呼び出しメソッド名に使われている動詞以外の単語．ただし，1 文字のものは除外する．

呼び出しているメソッドについては名前のみを考慮し，呼び出しているインスタンスの名前や型，呼び出しているメソッドの引数の数や型などは紐付けて考えない．また，型のジェネリクス宣言は考慮しないものとする．たとえば，List<String> は List として扱う．

トランザクションの例を図 1 に示す．この図の (a) のソースコードには，containsName，setSize という 2 つのメソッドが含まれており，containsName からは (b) が，setSize からは (c) が各メソッドのトランザクションとしてそれぞれ得られる．

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う．ルールを抽出する際に，6 つの条件をつけた．メソッド本体の内容に対してメソッド名の動詞を推薦するという目的のため，以下の 2 つの条件を満たすルールを抽出した．

1) 条件部がメソッド本体の動作を表す要素からなるもの

2) 帰結部にメソッド名の動詞のみが含まれているもの

[*1] http://wordnet.princeton.edu/

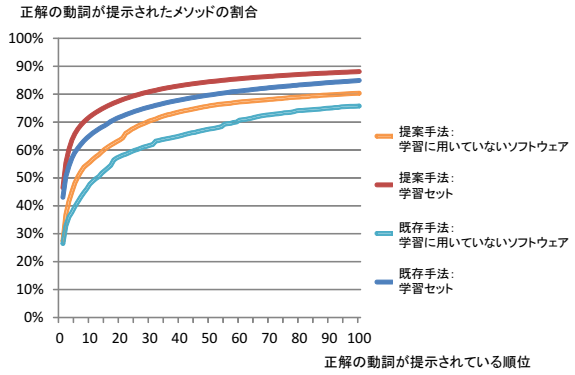

図 2 提案手法と既存手法の比較

さらに，より意味のあるルールを抽出するために，以下の 4 つの条件を満たすルールを抽出した．多くのメソッドで使われているルールを抽出するために条件 3 を，出現頻度が高い動詞が常に上位に提示されることを防ぐために条件 4 を，過剰適合を避けるために条件 5 を設定した．また，返り値が void であるメソッドが非常に多いため，条件 6 を設定した．

3) Support 値が 100 以上のもの
4) Lift 値が 1 以上のもの
5) 条件部の要素の数が 4 以下のもの
6) 条件部が「返り値の型 : void」のみでないもの

3.2 ステップ 2: 動詞の推薦

ステップ 1 で抽出したルールを用いて，対象のメソッドに対して動詞の候補を推薦する．本手法では，既存手法 [6] における動詞の推薦手法と同じ手法を用いて動詞の推薦を行うため，詳細の説明は省略する．

4. 評価

本研究では，本手法が既存手法と比較してどのくらい精度がよくなったか，開発者は動詞の候補リストを提示されたとき適切な動詞を選択できるようになるかについてそれぞれ評価を行う．評価するに当たって，あるメソッドに**既につけられている名前に使われている動詞**を，そのメソッドに対する**適切なメソッド名の動詞**とみなした．

評価に用いるルールを抽出するための学習セットとなるソフトウェア集合として，Qualitas Corpas[*2](バージョン 20130901) に示されている 112 個のオープンソースソフトウェアのバイナリファイル集合を選択した．学習に用いた 112 個のソフトウェアの詳細については表 1 に示す．既存手法を用いたとき，学習セットに含まれる 1,162,132 個のメソッドから 304,770 個のルールを得た．また，本手法を用いたとき 2,947,148 個のルールを得た．

[*2] http://qualitascorpus.com/

4.1 既存手法との比較

本手法が，既存手法に比べてどのくらい精度が改善されたのかを調査するために，本手法，既存手法それぞれの手法を用いて，相関ルールマイニングの入力として与えた学習セット内および学習に用いていないソフトウェア内のメソッドに対して動詞を推薦し，いくつのメソッドに対して，適切な動詞を推薦リストの上位何位に提示できるかを数えた．学習セットに含まれていないソフトウェアとして，Hayase ら [11] が用いていたソフトウェアから BlueJ, OrderPortal, Saxon-HE, NeoDatis の 4 つのソフトウェアを選択した．選択した 4 つのソフトウェアについての詳細は表 1 に結果とともに示す．

本手法と既存手法についてそれぞれ適用した結果を表 1, 図 2 に示す．表 1 において，#Method はメソッド総数を，ドメインは#Analyzed は，評価に用いる対象のメソッドの数を表す．#Recom および Top5 は実験結果を表しており，#Recom は#Analyzed のメソッドに対して動詞の候補リストを提示したとき，適切なメソッド名の動詞を提示できた数および割合を表し，Top5 は，適切なメソッド名の動詞を推薦リストの上位 5 位以内に提示できた数および割合を表す．(P) は既存手法を用いたとき，(N) は本手法を用いたときの結果をそれぞれ表す．図 2 において，横軸は候補リスト内で正解の動詞が提示されている順位，縦軸は対象のメソッド全体に対して，その順位までに正解が提示できているメソッドの割合を表す．例えば，70%のメソッドに対しては正解の動詞を候補リストの上位 10 位までに提示できることがわかる．

結果として，既存手法では，学習セットに含まれないソフトウェアのメソッドのうち 83.1%のメソッドに対し適切な動詞を提示でき，39.7%のメソッドに対して候補リストの 5 位以内に正解の動詞を提示できることがわかった．一方で，本手法を用いたとき，86.8%のメソッドに対して適切な動詞を提示でき，48.2%のメソッドに対して候補リストの 5 位以内に正解の動詞を提示できた．

また，学習できていた動詞の数について比較を行った．既存手法では，学習セットにおいて動詞と判定された 3,053 個の動詞の 14.0%にあたる，428 個の動詞について学習できた．本手法においては，13.8%にあたる，421 個の動詞について学習できた．

提示できる動詞の種類の数は変わらないものの，学習セットそのものに対しても，学習に用いていないソフトウェアに対しても本手法の方がより多くのメソッドに対してより上位に正解が提示できたことから，本手法は既存手法よりも優れていると考える．

4.2 候補リストの有無による有効性の評価

開発者が，動詞の候補リストを提示されているとき，提示されていないときに比べて適切な動詞を選択できるよう

表 1　学習セットおよび学習に用いていない 4 つのオープンソースソフトウェアの概要

ソフトウェア	ドメイン	#Method	#Analyzed	#Recom(P)	Top5(P)	#Recom(N)	Top5(N)
Qualitas Corpus	(none)	4,568,647	1,162,132	1,047,177(90.1%)	687,694(59.1%)	1,061,385(91.3%)	765,011(65.8%)
BlueJ 3.1.4[*3]	desktop application	8,188	2,800	2,391(85.4%)	1,100(39.3%)	2,332(83.3%)	1,346(48.1%)
OrderPortal 10.05.01[*4]	web application	41,241	3,392	3,118(91.9%)	1,060(31.3%)	3,044(89.7%)	1,344(39.6%)
Saxon-HE 9.6.0.4[*5]	xml	16,701	6,226	4,782(76.8%)	2,639(42.4%)	5,369(86.3%)	3,207(51.5%)
NeoDatis 1.9.30.689[*6]	database	4,472	1,445	1,224(84.7%)	708(49.0%)	1,284(88.8%)	781(54.1%)
sum	(NONE)	70,602	13,563	11,515(83.1%)	5,507(39.7%)	12,029(86.8%)	6,678(48.2%)

になるかを評価するために，3 つのリサーチクエスチョンを設定した．以下のリサーチクエスチョンを解決するために，被験者実験を行った．この実験は，被験者に複数の課題の解答と，アンケートの回答を行ってもらうものである．

RQ1: 動詞の候補リストがあるとき，開発者は適切なメソッド名の動詞を選択できるか

RQ2: 候補リスト内に正解が含まれている順位によって開発者の動詞の選び方にどのような傾向があるか

RQ3: 候補リストがあるとき，開発者が動詞を選ぶのにかかる時間が速くなるか

4.2.1 実験方法

提案手法を用いて開発者は適切なメソッド名の動詞を選択できるかどうかを評価するために，メソッド名が記述されていないソースコードを読んでそのメソッド本体の動作全体を表すような動詞を考えるという課題を複数の被験者に行ってもらう形式で行った．被験者は提案手法によって動詞の候補リストが提示されている課題と提示されていない課題の両方を実施する．このとき，動詞の候補は上位 5 位をリストで提示した．すべての課題の解答後に，開発経験等を問うアンケートを記入してもらった．

4.2.1.1 課題作成

課題の多様性を確保するため，複数のソフトウェアから課題となるメソッドを選択した．被験者の負担が大きくなりすぎないように，バイトコード命令が対応している行数が 5 行以上かつ 15 行以下のメソッドから選んだ．また，既につけられているメソッド名の目的語による影響を排除するため，メソッドの名前に動詞 1 語のみが付けられているメソッドを選んだ．また，課題全体で，提案手法が正解の動詞を提示する順位が同じになる課題が同じ数ずつになるように課題を選択した．選んだメソッド 1 つにつき，動詞の候補リストを提示する課題と提示しない課題を作成する．

課題 1 つにつき被験者に与えられるメソッドに関する情報は，メソッドが定義されているクラス名・親クラス名およびインタフェース名，メソッド本体でアクセスしているフィールドの定義，メソッド名を空欄としたメソッド 1 つである．パッケージのインポート文や，メソッド本体でアクセスされていないフィールドの定義，課題となるメソッド以外のメソッド，課題となるメソッド内外に記述されているコメントは削除する．これは，課題となるメソッド本体の動作を表す動詞を推測することを目的としている課題であり，他の場所で課題のメソッドが使用されている部分や課題のメソッドが説明されている箇所を探すものではないからである．メソッド内のコメントを削除するのは，課題ごとのコメントの有無が結果に影響を与える可能性を排除するためである．

表 1 に示した学習に用いていない 4 つのソフトウェアからそれぞれ 3 つずつ，計 12 のメソッドを選択した．提案手法が正解の動詞を 1 位，3 位，5 位に提示できているメソッド，提案手法が正解の動詞を上位 5 位以内に提示できていないメソッドがそれぞれ 3 つずつになるように選んだ．

4.2.1.2 被験者

被験者は，大阪大学基礎工学部情報科学科の学部 4 年生 3 人，および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修士課程の大学院生 9 人の計 12 人である．いずれもソフトウェア工学講座に所属している．実験に際し，開発経験を問うアンケートを行っている．被験者は，アンケート上で，Java プログラミングについて最短 1 年，最長 5 年の経験があると申告している．

4.2.1.3 課題の割り当て

被験者 1 人につき，8 つの課題を割り当てる．作成した 12 の課題を各プロジェクトから 1 つずつ，提案手法が正解の動詞を提示する位置がそれぞれ異なるように 4 つずつ 3 セットに分けた．1 人の被験者には 2 セットの課題を均等になるように割り当てる．被験者はそのうち 1 セットを手法による動詞の候補リストがある状態で，もう片方を動詞の候補リストがない状態で解く．被験者 12 人のうち，6 人は先に動詞の候補リストがある 1 セットを，残り 6 人は先に動詞の候補リストがない 1 セットを解いてもらった．セット内での課題の順番は，同一順序で課題を割り当てられている被験者はいないように割り当てた．

4.2.1.4 実施手順

5 分間の説明ののち，課題 8 問とアンケートを記入してもらう．先に動詞の候補リストがある課題を解く被験者と先に動詞の候補リストがない課題を解く被験者で別々に実施を行った．被験者しかいない部屋に集まってもらい，その場で実験アンケートの用紙を配付・回収を行う形式で行った．各課題を指定した順番に，後戻りして解答の修正

[*3] http://www.bluej.org/
[*4] https://launchpad.net/orderportal
[*5] http://saxon.sourceforge.net/
[*6] http://sourceforge.net/projects/neodatis-odb/

表 2 正解数

	正解	不正解	合計
候補リストあり	28	20	48
候補リストなし	25	23	48
合計	48	48	96

はせずに回答してもらった．時間制限は特に定めなかったが，各課題にどのくらい時間をかけているかを確認できるようにカメラを設置した．用紙 1 枚につき 1 つの課題を記載することで，ページをめくってから次のページをめくるまでをその課題にかかった時間とした．

4.2.2 RQ1: 動詞の候補リストがあるとき，開発者は適切なメソッド名の動詞を選択できるか

本手法によって推薦リストが提示されていた場合，提示されていないときに比べて，より適切な命名ができるかについて，分析を行った．まず，手法による動詞の候補リストがある場合とない場合で正解数にどのくらい差があるのかを調べた．手法がある場合とない場合の正解数を表 2 に示す．候補リストがある場合，ない場合と比べて，開発者が正解を選んでいる数が多いことがわかる．この差が有意であるかを確かめるために，統計検定を行った．G*Power3[*7] を用いてカイ二乗検定における検定力の事前分析を行ったところ，必要標本数が 88 であり，標本数が一定数以上であったためカイ二乗検定を行った．

帰無仮説 H_1，対立仮説 H_1' は以下のように表せる．

H_1 動詞の候補リストがあるときの正解数と，候補リストがない場合の正解数に差はない．

H_1' 動詞の候補リストがあるときの正解数は，候補リストがない場合の正解数より多い．

このとき，$p = 0.54$ となり，有意水準 $\alpha = 0.05$ のとき，$p \not< \alpha$ より，帰無仮説を棄却できない．よって，動詞の候補リストがある場合とない場合で，有意な差はなかった．

4.2.3 RQ2: 候補リスト内に正解が含まれている順位によって開発者の動詞の選び方にどのような傾向があるか

動詞の候補リストがある場合，候補リスト内に正解が含まれている順位によって正解数や解答の仕方に傾向があるかを分析した．開発者は正解が提示されている順位に関わらず，正解を選択できるのかについて分析した．もし開発者が提示されている順位に関わらず正解を選択できるならば，候補リスト内に正解が含まれていれば候補リスト内での順位を強く考慮する必要はなくなると考えたからである．

正解が提示されている順位ごとの正解数を表 4 に示す．例えば，候補リストが提示されたときに正解が 5 位に提示される課題での，候補リストがない場合に正解が選ばれた課題は，4 つであることが読み取れる．候補リストの 1 位に正解が提示される課題については，候補リストが提示されていない場合でも被験者が正解の動詞を解答しているのに対して，3 位，5 位に正解が提示される課題については，候補リストがある場合の方が正解数が多く，正解が提示されない課題については，候補リストがない場合の方が正解数が多いことがわかる．従って，開発者は 1 位に提示されるようなメソッドは，候補リストがなくても開発者は一定の名前を付けられると考えられる．候補リスト内の 3 位や 5 位に提示されるようなメソッドは，候補内に正解が提示されていれば，62%の人は解の動詞が適切だと判断して名前を付けることができている．ここから，候補リストの上位 5 位以内に正解を提示していれば，開発者は正解の動詞を選択できると考えられる．

表 3 正解が提示されている順位と開発者が解答した動詞が提示されている順位のピポットテーブル

	1 位	2 位	3 位	4 位	5 位	リスト外	正解数
1 位	11	0	1	0	0	0	11
3 位	0	2	8	0	1	1	8
5 位	0	1	4	0	7	0	7
正解なし	2	0	1	0	1	8	2

また，表 3 に，正解が提示されている順位が同じ課題ごとに，開発者が何位に提示されている動詞を解答しているかを示す．表において，列は開発者が選んだ動詞が提示されている順位，行は正解の動詞が提示されている順位を表す．例えば，1 位に正解が提示されていて，開発者が 1 位に提示されている動詞を解答していた課題が 11 個あったことを表す．表 3 から，候補リストの中に正解がない場合でも，候補リストの中から選んでいるのではなく，リスト外の動詞を用いて解答していることから，正解ではない候補がリストに混ざっていても，開発者はそれらが正解の動詞ではないと判断できていると考えられる．

候補リストに正解が含まれているとき，候補リストがある方が有意に正解数が多いのかを確かめるために，統計検定を行った．表 5 に，候補リスト内に正解が提示されている課題について，候補リストがある場合とない場合の正解数を示す．カイ二乗検定の事前分析により得られた必要な標本数より少なかったため，フィッシャーの正確確率検定を行った．帰無仮説 H_1，対立仮説 H_1' は以下のように表せる．

H_1 動詞の候補リストがあるときの正解数と，候補リストがない場合の正解数に差はない．

H_1' 動詞の候補リストがあるときの正解数は，候補リストがない場合の正解数より多い．

このとき，$p = 0.045$ となり，有意水準 $\alpha = 0.05$ のとき，$p \not< \alpha$ より，帰無仮説が棄却される．よって，動詞の候補リストがある場合とない場合で，有意な差があったといえる．

[*7] http://www.gpower.hhu.de/

表 4　正解が提示されている順位ごとの正解数

正解が提示されている順位	候補リストあり		候補リストなし	
	正解	不正解	正解	不正解
1 位	11	1	10	2
3 位	8	4	4	8
5 位	7	5	4	8
正解なし	2	10	7	5

表 5　候補リスト内に正解が含まれている場合の正解数

	正解	不正解	合計
候補リストあり	26	10	36
候補リストなし	18	18	36
合計	44	28	72

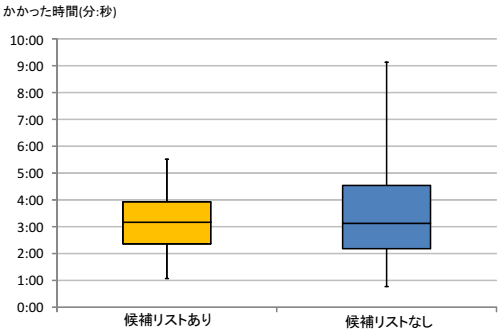

図 3　候補リストがあるときとないときの課題にかかった時間

表 6　課題にかかった時間から得られる統計的な指標値

	平均値	中央値	標準偏差	変動係数
候補リストあり	3 分 10 秒	3 分 7 秒	1 分 12 秒	0.38
候補リストなし	3 分 34 秒	3 分 5 秒	1 分 52 秒	0.53

4.2.4　RQ3: 候補リストがあるとき，開発者が動詞を選ぶのにかかる時間が速くなるか

候補リストの提示がある場合と無い場合で，解答にかかる時間に差があるのかを調査した．候補リストが提示されている方が，提示されていない場合よりも早く動詞を選択できるのではないかと仮定した．

候補リストがある場合とない場合の解答時間の分布を図 3 に，統計的な指標値を表 6 に示す．図 3 において，横軸が候補リストの有無を表し，縦軸が課題にかかった時間を表す．候補リストの提示がある場合の平均解答時間については，候補リストがある場合の方が速かった一方で，中央値は候補リストの提示があるとない場合で差がなかった．また，候補リストがある場合，ない場合と比較して変動係数が小さいことから，解答時間のばらつきが少ないことがわかる．ここから，被験者が動詞の選択に悩んだときに，候補リストがすばやく動詞を選択するための助けになっていたのではないかと考えられる．

4.2.5　考察

結果として，候補リストがある場合の方が正解数は多かったが，有意差はみられなかったことがわかった．また候補リストがある場合の方が解答にかかる時間のばらつきが少ない傾向にあることがわかった．さらに，候補リストに正解が提示されているメソッドについては，被験者が正解の動詞を選択できることがわかった．

候補リストの有無によって正解数に有意差が見られなかった原因として，被験者の所属がソフトウェア工学講座であったことが考えられる．課題後のアンケートでは，「後から読んでわかる名前を付けることを心がけているかどうか」という質問に対し，12 人中 11 人が心がけていると回答している．また，「命名について心がけていることはあるか」という自由記述による質問に対し，「役割がわかりやすいようにする」といった回答や，「メソッドシグネチャだけで動作がわかるように心がけている」といった回答もあった．ソフトウェア工学は，品質の高いソフトウェアを低コストで期限内に開発し，効率良く保守するための技術を扱う学問分野であるため，ソフトウェアの品質についての研究をしている人たちは，一般の開発者に比べて保守性や命名に対する意識が高いのではないかと考えられる．

正解数に有意な差が見られなかったものの，候補リストがある場合の方が，安定した速度で解答できることがわかった．また，候補リスト内に正解が提示されている場合，候補リストがある場合の方が正解数が多かったことから，手法により動詞の候補リストの上位 5 位以内に常に正解を提示できれば，正解数が上がるのではないかと考える．

5.　妥当性の脅威

ルールの抽出には，特定のソフトウェア集合を用いた．多様なドメインのソフトウェアが集められている既存のソフトウェア集合を利用したが，意図しない偏りがあるかもしれない．そのため，抽出に用いるソースファイル集合を変更した場合，結果が大きく変わるかもしれない．

ルールを抽出するためのデータはバイトコードから取得した．同一もしくはバージョンが異なるソフトウェアがライブラリとして複数のソフトウェアでインポートされていたり，オープンソースのコードをプロジェクトの中にコピーアンドペーストなどで取り込んだりしている場合，同一のメソッドを複数学習に用いている可能性がある．しかし，多くのソフトウェアで使われているライブラリの命名ルールは，一貫した推薦になりうると考えているため，大きな影響はないと考える．

動詞を判定するために，WordNet の辞書を用いた．プログラミング言語で使われている単語は自然言語とは異なるが，WordNet は自然言語の辞書であるため，異なる語が動詞として判定されている可能性がある．Arnaudova らも WordNet を用いているが，WordNet がソフトウェアの分野には適していないものの，辞書の入れ替えは簡単でありすぐに変更できると主張している [12]．

本研究では，動詞から始まるメソッド名のみを対象にした．あるガイドラインでは，動詞から始まるメソッド名は

対象のインスタンス内で値を書き込むメソッドに使うべきであり，値を読み取って取得するだけのメソッドに対しては動詞から始まる名前を付けるべきではないと主張している [13]．本研究では，学習と評価の両方の対象をメソッド名の最初に動詞が使われているメソッドのみに限定しているため，影響は少ないと考える．また，Yu らも同様にメソッド名の最初に出現する動詞をメソッド名の動詞として扱っている [10]．

手法を評価するため，オープンソースソフトウェア内のメソッドに対して，既に使われている動詞がそのメソッド名の適切な動詞であると仮定し，類語などは考慮せずに正解をただ 1 つに定めた．この仮定は証明されていないが，Yu らも同様の仮定を用いているため [10]，1 つの評価基準としては妥当であると考える．

被験者実験の課題は，著者が手作業で選択した．なるべく偏りがないように動作が異なるメソッドを選択したが，意図しない偏りが生じている可能性がある．

候補リスト内に正解がある場合の方が，正解数が有意に多いかを確かめるために，統計検定を行った．カイ二乗検定に最低限必要だと計算された標本数に満たなかったためフィッシャーの正確確率検定を用いた．標本数を十分に増やして検定を行うと結果が変わるかもしれない．

被験者実験における課題にかかった時間についてはカメラで撮影した映像から，筆者が計測したが，ページをめくる早さなどにより，計測に誤差が生じている可能性がある．しかし，課題にかかる時間が分単位であるため，数秒の誤差による影響は少ないと考えられる．

6. 結論

本研究では，メソッド本体に出現する単語および型とメソッド名に用いる動詞の相関ルールを利用して，メソッド名に用いる動詞の候補リストを提示する既存手法を改良した．また，改良した手法が既存手法よりも適切な候補リストを提示できるかどうかを評価し，改良した手法を用いて，候補リストが提示されたときに開発者がより適切な命名をできるかどうかについて調査した．その結果，本手法はルールの抽出に用いていないソフトウェアに対して 86.8%のメソッドに対して正解の動詞を提示でき，48.2%のメソッドに対しては候補リストの上位 5 位以内に提示できており，既存手法よりも適切な動詞を候補リストの上位に提示できることを確かめた．開発者が候補リストを提示されている場合，提示されていないときよりも正解数が多かったものの，有意差はなかった．一方で，開発者は候補リストに正解が提示されているとき，候補リストがないときよりも被験者は正解の動詞を選択できることがわかった．

今後の課題として，より多くのメソッドに対して候補リストの上位 5 位以内に正解を提示できるように提案手法を改良する必要があると考える．特に，上位 5 位以内に正解が出現するように候補リストの並べ方を改善することが必要だと考える．また，本手法が実際に開発現場で使えるのかどうかを評価する必要がある．本研究における実験では，被験者が知らないメソッドを課題とし，候補リストがある場合の方が適切な命名ができるのかを評価した．しかし，実際の開発環境では開発者は周辺のメソッドとの呼び出し関係やそのメソッドの役割について知っていると考えられるので，実際の環境での効果を評価する必要がある．

謝辞 本研究は，科研費 (2522003, 26280021) の助成を得たものである．

参考文献

[1] Lawrie, D., Morrell, C., Feild, H. and Binkley, D.: What's in a Name? A Study of Identifiers, *Proc. ICPC*, pp. 3–12 (2006).
[2] De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A. and Panichella, S.: Using IR methods for labeling source code artifacts: Is it worthwhile?, *Proc. ICPC*, pp. 193–202 (2012).
[3] McConnell, S.: *Code Complete, Second Edition*, Microsoft Press (2004).
[4] Oracle: Code Conventions for the Java TM Programming Language (1997).
[5] Høst, E. W. and Østvold, B. M.: The Programmer's Lexicon, Volume I: The Verbs, *Proc. SCAM*, pp. 193–202 (2007).
[6] Kashiwabara, Y., Onizuka, Y., Ishio, T., Hayase, Y., Yamamoto, T. and Inoue, K.: Recommending Verbs for Rename Method using Association Rule Mining, *Proc. CSMR-WCRE* (2014).
[7] Agrawal, R., Imielinski, T. and Swami, A.: Mining Association Rules between Sets of Items in Large Databases, *Proc. SIGMOD*, pp. 207–216 (1993).
[8] Høst, E. W. and Østvold, B. M.: Debugging Method Names, *Proc. ECOOP*, pp. 294–317 (2009).
[9] Karlsen, E. K., Høst, E. W. and Østvold, B. M.: Finding and fixing Java naming bugs with the Lancelot Eclipse plugin, *Proc. PEPM*, pp. 35–38 (2012).
[10] Yu, S., Zhang, R. and Guan, J.: Properly and Automatically Naming Java Methods: A Machine Learning Based Approach, *Advanced Data Mining and Applications*, Vol. 7713, Springer Berlin Heidelberg, pp. 235–246 (2012).
[11] Hayase, Y., Kashima, Y., Manabe, Y. and Inoue, K.: Building Domain Specific Dictionaries of Verb-Object Relation from Source Code, *Proc. CSMR*, pp. 93–100 (2011).
[12] Arnaoudova, V., Di Penta, M., Antoniol, G. and Gueheneuc, Y.-G.: A New Family of Software Anti-patterns: Linguistic Anti-patterns, *Proc. CSMR*, pp. 187–196 (2013).
[13] Green, R. and Ledgard, H.: Coding Guidelines: Finding the Art in the Science, *Communications of the ACM*, Vol. 54, pp. 57–63 (2011).