修士学位論文

題目

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

指導教員
井上 克郎 教授

報告者
柏原 由紀

平成 27 年 2 月 6 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 26 年度 修士学位論文

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

柏原 由紀

内容梗概

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．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．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．

そこで本研究では，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．相関ルールマイニング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．本研究では評価実験として，1) 提案手法が適切な候補リストを提示できるか，2) 候補リストが提示されている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評価を行った．その結果，1) ルールを抽出していないソフトウェアについて， 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ることがわかった．また，2) 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ていることがわかった．

主な用語

プログラム理解
メソッド名の命名支援
相関ルール

目 次

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]．開発者は保守作業の大半の時間をソースコードの読解に費やしており [2]，プログラム理解に時間がかかることは保守コストの増大にもつながっている．また開発者は，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 [3]．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 [4][5] で主張されていることからも，ソースコード中で適切な識別子を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 [4] が，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 [6] が，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．

既存研究では，メソッドに対する適切な命名を支援する手法が提案されている．Høst らはメソッド名の動詞に着目し，メソッドの内容に対して非常に悪い動詞を命名に用いているメソッドを自動で検出し，その修正の候補を提示する手法を提案している [7]．Høst らの手法は，メソッドの内容と命名に用いる動詞の対応関係を一対一関係のルールで表しているため，適用できるメソッドが非常に少ないという問題がある．Yu らはメソッドに対して，機械学習により自動でそのメソッドに対して最も適切だと考えられ動詞を命名する手法を提案した [8]．Yu らはサポートベクターマシン (SVM) を用いており，適用できるメソッドが多く，扱う動詞の種類も多い．しかし，彼らの手法はメソッドを自動で命名することを目的としているため 1 つのメソッドにつき 1 つの動詞しか提示することができず，もし開発者がその動詞を不適切だと判断した場合，開発者はそれ以上参考にできる情報がなくなってしまう．

そこで本研究では，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．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 [9] 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．

実験では，Java で記述されたソフトウェアを対象に 2 つの観点から手法を評価した．112 個のオープンソースソフトウェアから抽出したルールを用いて，1) 手法が適切な候補リスト

を提示できるか，2) 候補リストが提示されている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．評価の際，評価対象の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．その結果，1) ルールを抽出していないソフトウェアについて，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ることがわかった．また，2) 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ていることがわかった．

本稿における貢献は以下のとおりである．

- 既存ソフトウェア群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表すルールを自動で抽出する手法を提案した．
- 内容が記述されたメソッドに対してメソッド名に用いる動詞の候補をリストとして推薦する手法を提案した．
- 提案手法が候補リストの上位に適切な動詞を提示できていることを示した．
- 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適切な動詞を選択できていることを示した．

以降，2 章では背景について述べ，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　背景

本研究の手法の背景として，プログラミング言語の識別子について，一般的な識別子の命名方法と，オブジェクト指向プログラミング言語である Java の識別子に対する命名方法，そして Java の識別子の 1 つであるメソッドの命名について，順に説明する．次に，プログラミングにおける保守性を高めるリファクタリングについての概要と，リファクタリングの 1 つであるリネームリファクタリングについて説明する．そして現在提案されているいくつかのメソッドの命名に関する既存研究を説明し，最後に，提案手法で用いる相関ルールマイニングについて説明する．

2.1　識別子

本節では，一般的な識別子の記法を説明してから，Java に登場する識別子について識別子の種類ごとに命名の特徴を説明する．その後，本研究で着目する Java におけるメソッドの命名についてさらに詳しく述べる．

2.1.1　一般的な識別子の記法

プログラム中の識別子は 1 つの単語もしくは複数の単語の列を用いて命名されるが，識別子の中では，単語境界に空白文字を用いることができない．空白を用いずに複数の単語をひと綴りで表現する記法にはキャメルケース，パスカルケース，スネークケースなどがある．キャメルケースは単語の列の先頭文字を小文字に，単語境界となる単語の先頭文字を大文字で表して単語をつなげる記法である．一方，パスカルケースは単語の列の先頭文字および単語境界となる単語の先頭文字をどちらも大文字で表して単語をつなげる記法である．また，スネークケースは単語が大文字表記か小文字表記かにかかわらず，単語間をアンダースコアでつなげる記法である．「example」と「identifier」の 2 つの単語をつなげる場合，キャメルケースの記法では exampleIdentifier，パスカルケースの記法では ExampleIdentifier となり，スネークケースの記法では example_identifier となる．

2.1.2　Java における識別子の命名

Java の識別子には，クラス名，変数名，メソッド名がある．Java の命名規約 [1] によると，クラス名に分類される具体的な識別子にはクラス名とインタフェース名があり，一般にパスカルケースで表現した単語もしくは単語の列として命名される．これらの単語には，クラスの役割を表現しているような，略語ではない名詞が使われる．

[1] http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

変数は，フィールド変数やローカル変数などをさし，その名前は一般にキャメルケースで表現した 1 つ以上の単語の列として命名される．for 文などで使われるローカル変数がアルファベット 1 字の命名が許されることを除いて，役割を表現する単語もしくはその略語を用いて命名する．変数の中でも定数は，final 修飾子によってプログラム実行中に値が変化しない変数のことをさし，その名前はすべて大文字の単語をスネークケースで結合した 1 つ以上の単語の列として命名される．変数名と同様に，役割を表現する単語やその略語を用いて命名する．

メソッド名は一般的には小文字で始まる動詞を先頭単語としてキャメルケースで表現された 1 つ以上の単語の列で構成され，メソッド名全体で自然言語の動詞と目的語のような関係をもつことが多い [10]．また，メソッド名の目的語に用いられる単語の列は，メソッドの動作の対象となる識別子を用いることが多いと考えられている [4]．

2.1.3 Java におけるメソッドの命名

メソッドの命名に用いるべき単語については規約として定義されていないが，開発者の間で暗黙的に共通しているいくつかの動詞の使い方がある．フィールドへの Accessor メソッドには get および set の動詞を用いることは特に一般的であり，Eclipse[2] などの統合開発環境にも，標準機能として get および set の動詞を用いた Accessor メソッドを補完する機能が提供されている．また，ブール値を返すメソッドは is，contains，can といった動詞を使うこと，JUnit によるテストメソッドは test の動詞から始まるといった使い方については，統合開発環境 IntelliJ[3] には，命名の自動チェック機能として実装されている．

特殊なメソッドの命名パターンとして，目的語-動詞の順に並べて命名することや動詞を使わずに命名することもある．目的語-動詞の順に単語を並べたメソッド名は，マウスの動きやボタンのクリックなどを監視して，動作が発生したときに呼び出されるメソッドの名前としてよく出現しており，例として，java.awt.event.ActionListener インタフェースの actionPerformed メソッドがある．また，動詞を使わない命名の例としては，java.lang.Object クラスの toString メソッドなどの前置詞から始まるメソッドが挙げられる．

2.2 リファクタリング

リファクタリングとは，ソフトウェアの品質を向上させるためにもっともよく使われる重要な技術の 1 つである [11][12]．外部から見える振る舞いを保ちつつ，プログラムの理解や修正が簡単になるように内部構造を変更することであり，長いメソッドの一部分を新たなメソッドとすることで処理の分割を行うメソッド抽出リファクタリング，識別子を変更するリ

[2] https://eclipse.org/
[3] https://www.jetbrains.com/idea/

ネームリファクタリングなどがある．リファクタリングはWilliamらによって以下のように定義されている [13].

> The process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure.

手作業でリファクタリングを行うと，バグを埋め込んでしまいやすくなるが，内部構造の変更によって外部の振る舞いに影響を与えないように，自動でリファクタリングを行うためのツールが開発されている．IntelliJやEclipseといった統合開発環境には標準プラグインとして実装されている．Bavotaらは，リファクタリングによってバグを埋め込んでしまう確率は低く，開発者は手動でリファクタリングを行うよりも，ツールを用いてリファクタリングを行うべきだと主張している [14]．Murphy-Hillらは，リファクタリングをするときの開発者の振る舞いについて調査しており [15]，実際に開発者はツールを用いてリファクタリングを行っていることがわかっている．

2.2.1 リネームリファクタリング

リネームリファクタリングは，識別子をよりわかりやすい名前に変更するリファクタリングであり，最もよく行われているリファクタリングである [16]．Arnaoudovaらは，商用ソフトウェアもしくはオープンソースソフトウェアの開発者計71人に対する調査によって，39%が週数回からほとんど毎日，46%が月数回のリネームリファクタリングを行うということを示した [17]．さらにArnaoudovaらは，ソフトウェアの各バージョン間で，どのようなリネームリファクタリングが行われているか調査しており，他の識別子よりもメソッドのリネームリファクタリングが頻繁に行われていることを示した．ここから，メソッドの命名支援が重要であると考える．

2.3 メソッドの命名支援

メソッドを含む不適切な識別子の命名パターンについては，AbebeらやArnaoudovaらによって調査されている．Abebeらは，主に変数を対象に，識別子の命名における「lexicon bad smells」を定義した [18].「lexicon bad smells」は，ある識別子の命名そのものや，2つ以上の識別子の間に存在する潜在的な問題のことである．例えば,「Odd grammatical structure」として，クラス名が動詞のみ，メソッド名が名詞のみで命名されている場合などが定義されている．Arnaoudovaらは，メソッドと変数それぞれに着目して，識別子の命名における「anti-patterns」を定義した [19]．メソッドの命名における「anti-patterns」については，メソッドシグネチャやメソッド本体に出現する識別子，メソッド内外に記述されているコメントの間に存在する潜在的な問題のことである．例えば,「“Get” more than an accessor」と

して，その名前が get から始まるメソッドの中でフィールドの値を返す以外の処理を行っている場合などが定義されている．Abebe らや Arnaoudova らは不適切な命名のパターンを発見するツールも作成している．しかし，どちらの研究においても不適切な命名パターンについては言及しているが，適切な命名パターンについては言及していない．

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 [7]．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 [20]．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 [8]．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．目的語についても SVM を利用して提示しており，メソッド内に出現する識別子が目的語になりやすいという点に着目し，メソッド内に出現する識別子がそのメソッド名の目的語として適切かどうかをブール値で提示している．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．

Suzuki らは，メソッド名が複数の単語から成ることに着目し，メソッド名全体を命名するための支援を行う手法を提案した [21]．N-gram 法を用いてメソッド名を分割し，ある単語が出現したとき，次にどの単語が続くかを確率モデルにより推測する．彼らの手法は，メソッド名の最初の単語を推薦できないという問題がある．本手法と組み合わせることで，メソッド名全体の命名支援ができるのではないかと考えている．

2.4 相関ルールマイニング

相関ルールマイニングとは，大量のアイテム集合 (トランザクション) の集合を入力として，あるアイテムがトランザクションに出現したときに，別のアイテムもまた同じトランザクションに出現する可能性が高いという関係を抽出する手法である．相関ルールマイニングによって抽出されるアイテム間の関係を相関ルールという．相関ルールをアイテム集合 X，Y を用いて，式 (1) のように表す．このとき，式 (1) における X を条件部，Y を帰結部といい，X，Y は入力に与えたトランザクションの部分集合となる．

$$X \to Y \tag{1}$$

入力として用いたトランザクションの集合に対して抽出した相関ルールがどのくらい強い相関を持っているかを評価する指標として，Support 値，Confidence 値，Lift 値という 3 つの値がある．一般に Support 値は，すべてのトランザクションに対する条件部と帰結部がともに部分集合となるトランザクションの数の割合であるが，本研究では，条件部と帰結部がともに部分集合となるトランザクションの数そのものを Support 値とみなす．Confidence 値は，条件部が部分集合となっているトランザクションのうち，帰結部もまた部分集合となっているトランザクションの割合を表す．Lift 値は，条件部が部分集合となっているトランザクションの数に対する条件部と帰結部がともに部分集合となっているトランザクションの数の割合が，全トランザクションに対する帰結部が部分集合になっているトランザクションの数の割合の何倍であるかを表した値である．Support 値が高いとその相関ルールを満たすトランザクションの数が多いことがわかり，Confidence 値が高いと，条件部が部分集合となるトランザクションは帰結部のアイテムを含んでいることが多いことがわかる．Lift 値に関しては，Lift 値が低いとき，条件部が同時に含まれているかどうかに関わらず，帰結部が部分集合となっているトランザクションの数自体が多いことを表しており，一般に Lift 値が 1 より大きいとき有用なルールだといわれている．相関ルールマイニングでは，これらの値がある閾値以上であるときに相関があるとみなすが，この閾値は相関ルールマイニングを行うときに自由に決めることができる．

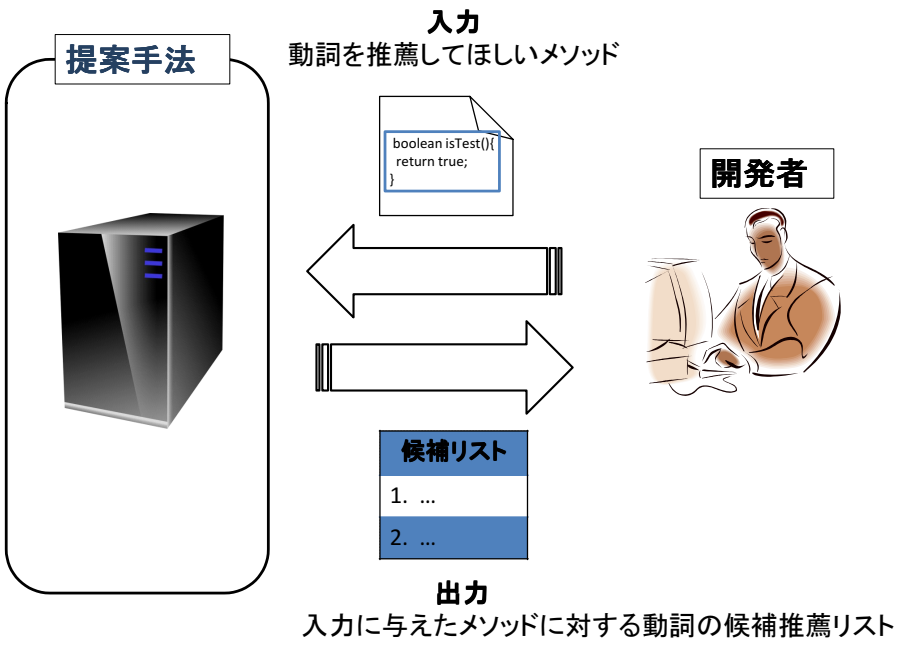

図 1: 利用する開発者から見える手法の入出力

3 提案手法

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．利用想定を図 1 に示す．開発者はソフトウェアの実装を終えたあと，ソフトウェアの保守性を高めるための保守行程に入る前に手法を用いてリネームリファクタリングを行うとする．本手法は，開発者が記述済みのメソッドを入力として与えたとき，そのメソッドに対してメソッド名の動詞の候補リストを作成し，開発者に提示する．開発者は，提示されたリストを参考にして，メソッド名の命名や変更を行う．

本手法の概要を，図 2 に示す．本手法は大きく分けて 2 ステップに分かれている．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 である．ステップ 1 は，前処理として一度だけ行い，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．各ステップについて順に説明する．

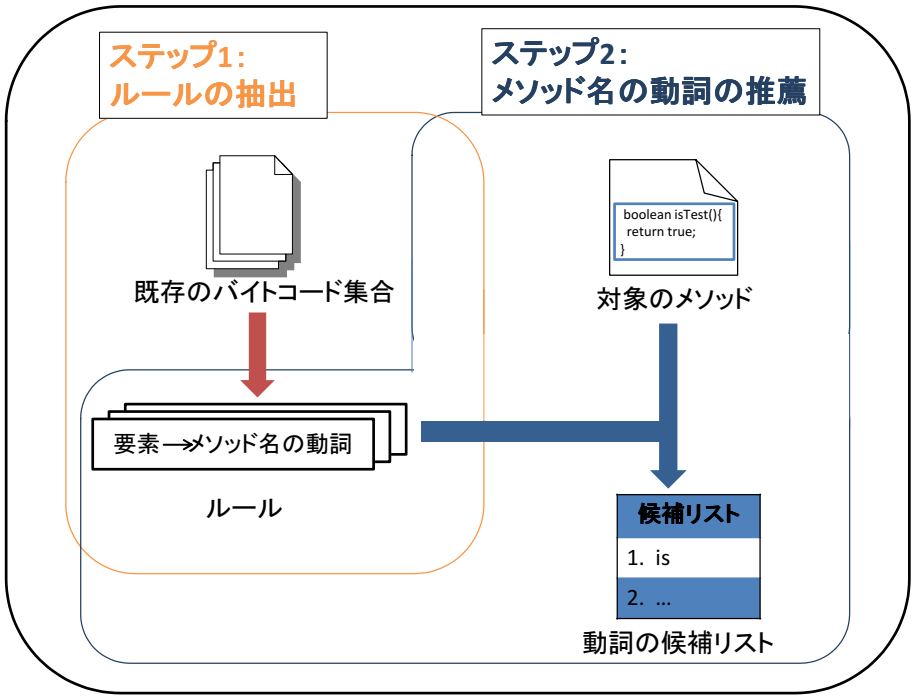

図 2: 手法の概要図

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．本ステップは 3 つのサブステップから成る．図 3 にステップ 1 の概要を示す．

サブステップ 1-1 ソフトウェア集合から学習に用いるメソッド集合を取得する

サブステップ 1-2 メソッド集合からトランザクションの集合を取得する

サブステップ 1-3 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適用する

以降の節で各サブステップについて順に説明する．

3.1.1 サブステップ 1-1: ソフトウェア集合から学習に用いるメソッド集合を取得する

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，メソッド集合を取得する．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 においてキャメルケース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

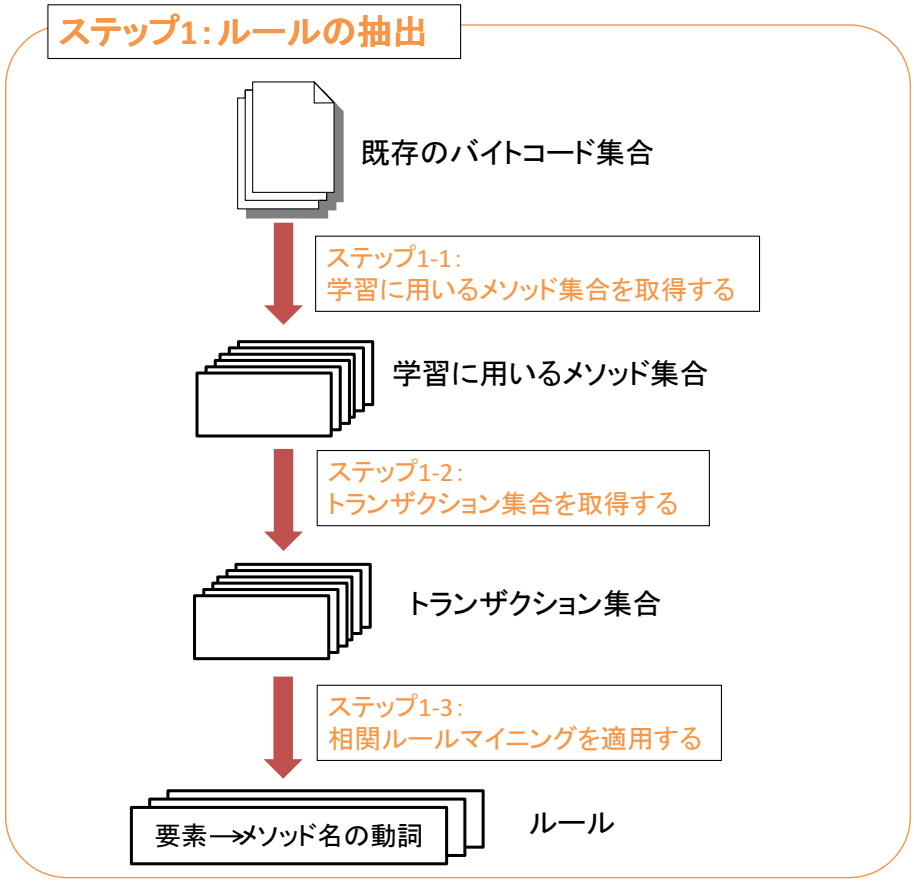

図 3: ステップ 1 の概要図

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めである．スネークケース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，メソッド名が動詞から始まることが多いため，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 [8].

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．UML や XML 等の大文字で記述することが一般的な単語が使われることがあるが，その場合 1 つのメソッド名の中で大文字が連続して出現することがある．大文字が 2 つ以上連続したあと小文字が出現する場合は，最初に出現した大文字から大文字の最後の 2 文字目までを 1 つの単語，大文字の最後の 1 文字を次の単語の最初の 1 文字として分割を行った．例えば,「exampleIdentifier」の場合「example」と「Identifier」に分解し,「exampleCASEIdentifier」の場合「example」,「CASE」そして「Identifier」に分解し,「exampleCASE_IDENTIFIER」の場合「example」,「CASE」,「IDENTIFIER」とそれぞれ分割する．

動詞の判定には品詞解析ができる自然言語処理ツールを利用しており，実装では OpenNLP[4] を用いた．品詞を判定するために用いる辞書として，WordNet[5] 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 が動詞として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 が動詞として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．

[4]https://opennlp.apache.org/
[5]http://wordnet.princeton.edu/

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．

さらに，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．

3.1.2 サブステップ 1-2: メソッド集合からトランザクションの集合を取得する

サブステップ 1-1 で選択したメソッド集合を，相関ルールマイニングの入力となるようにトランザクション集合に変換する．メソッド集合に含まれる各メソッドをトランザクションに変換することで，トランザクション集合を得る．

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 4 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．

```
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 4: トランザクションの取得例

3.1.3 サブステップ 1-3: 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適用する

サブステップ 1-2 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．

条件 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

条件 2 帰結部にメソッド名の動詞のみが含まれているもの

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．

条件 3 Support 値が 100 以上のもの

条件 4 Lift 値が 1 以上のもの

条件 5 条件部の要素の数が 4 以下のもの

条件 6 条件部の要素が「返り値の型 : void」のみでないもの

3.2 ステップ 2: メソッド名の動詞の推薦

本ステップでは，ステップ 1 で抽出したルールと開発者が指定したメソッド本体の内容を用いて，メソッド名の動詞の候補リストを生成する．ステップ 2 の概要を図 5 に表す．本ステップは 2 つのサブステップから成る．

サブステップ 2-1 対象のメソッドへの動詞の推薦に用いるルールを選択する

サブステップ 2-2 選択したルールからメソッド名の動詞の推薦リストを作成する

以降，各サブステップについて順に説明する．

3.2.1 サブステップ 2-1: 対象のメソッドへの動詞の推薦に用いるルールを選択する

メソッド名に用いる動詞の候補リストを提示したいメソッドのトランザクションを検索キーとして，ステップ 1 で抽出したルール群から推薦に用いるルールのみを選択する．命名の候補リストを提示したい 1 つのメソッドから，トランザクションを抽出し，ステップ 1 で抽出したルール群から，条件部が対象メソッドから得られるトランザクションの部分集合となるルールをすべて抽出する．その後，条件部がトランザクションの部分集合のうち，同じ

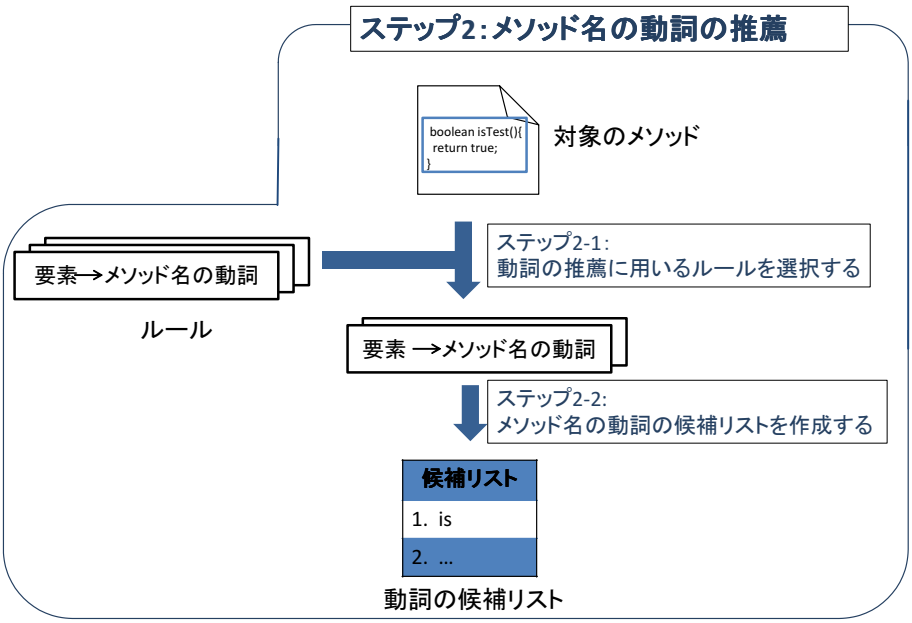

図 5: ステップ 2 の概要図

動詞を帰結部に持つルールごとに，Confidence 値が一番高いルールのみを選択し，他のルールは除外する．従って，残ったルールはそれぞれ異なる動詞を帰結部を持つ．

3.2.2 サブステップ 2-2: 選択したルールからメソッド名の動詞の推薦リストを作成する

サブステップ 2-1 で選択したルールから，メソッド名の動詞の候補リストを作成する．まず，選択したルールを各ルールの Confidence 値の降順に並び替える．各ルールの帰結部の動詞を，並び替えたルールと同じ順序でリストとして提示する．開発者はこの動詞のリストを参考にして，メソッド名の変更を行う．

4　評価

本研究では，以下の 2 つの観点で評価を行う．

評価 1) 提案手法は，適切な動詞の候補リストを提示できるか．

評価 2) 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．

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．Yu らも，同様の仮定をおいて彼らの手法を評価している [8]．

4.1　準備:ルールの抽出

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[6](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付録の表 11 に示す．学習セットに含まれる，かつ，3.1.1 節の条件を満たしている 1,162,132 個のメソッドからルールの抽出を行った結果，2,947,148 個のルールを得た．

4.2　評価 1) 提案手法は，適切な動詞の候補リストを提示できるか

提案手法が，適切な動詞の候補リストを提示できるかについて評価するために，2 つのリサーチクエスチョンを設定した．

RQ1: 提案手法が学習に用いたメソッド自体に対して適切な動詞を上位に提示できるか．

RQ2: 提案手法が学習に用いていないメソッドに対して適切な動詞を上位に提示できるか．

4.2.1　RQ1: 提案手法が学習に用いたメソッド自体に対して適切な動詞を上位に提示できているか

相関ルールマイニングの入力として与えた学習セットに対して，どのくらいルールが抽出されるかを調査した．学習セット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．

結果を表 1 および図 6 に示す．表 1 において，#Method はメソッド総数を，#Analyzed は，評価に用いる対象のメソッドの数を表す．#Recomm および Top5 は実験結果を表して

[6]http://qualitascorpus.com/

表 1: 学習セットの概要

対象ソフトウェア	#Method	#Analyzed	#Recomm	Top5
Qualitas Corpus	4,568,647	1,162,132	1,061,385(91.3%)	765,011(65.8%)

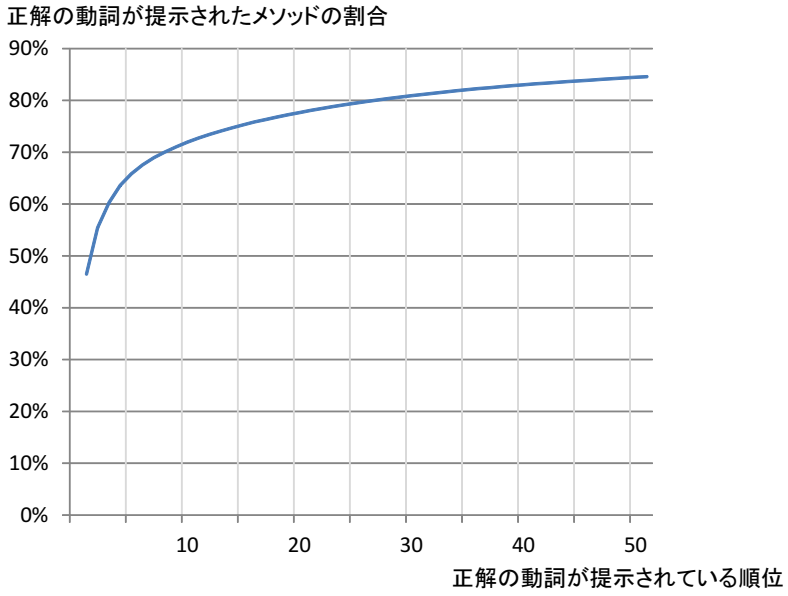

図 6: 学習セットに対して手法を適用した結果

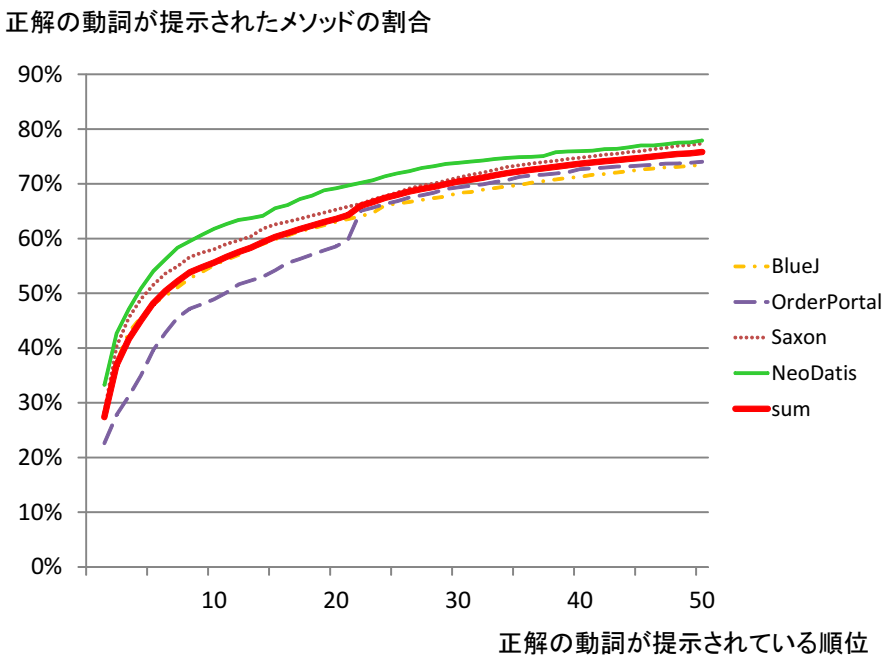

図 7: 学習に用いていないソフトウェアに対して手法を適用した結果

おり，#Analyzed のメソッドに対して提案手法を適用したときの，#Recomm は適切なメソッド名の動詞を提示できた数および割合を，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．図 6 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 位までに提示できていることがわかる．

結果として，学習に用いた 1,162,132 のメソッドのうち，91.3%のメソッドに対して適切な動詞を提示できた．また，65.8%のメソッドに対しては，上位 5 位に適切な動詞を提示できている．学習セットに対しては適切な動詞を推薦できていると考える．

また，どのくらいの種類の動詞について学習できていたのか調査した．学習セットにおいて動詞と判定された 3,053 個の動詞の 13.8%にあたる，421 個の動詞について学習できた．学習できている 421 個の動詞は，学習セットとして用いたメソッドの少なくとも 91.3%を占めていることから，よく使われる動詞については，学習できていると考える．

4.2.2 RQ2: 学習に用いていないメソッドに対して適切な動詞を提示できるか

学習セットに用いていないメソッドに対して，どのくらい適切な動詞を推薦できるかどうかを調査した．学習セットに用いていない 4 つのオープンソースソフトウェアのメソッドに

表 2: 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	#Recomm	Top5
BlueJ[7] (3.1.4)	desktop application	8,188	2,800	2,332(83.3%)	1,346(48.1%)
OrderPortal[8] (10.05.01)	web application	41,241	3,392	3,044(89.7%)	1,344(39.6%)
Saxon-HE[9] (9.6.0.4)	xml	16,701	6,226	5,369(86.3%)	3,207(51.5%)
NeoDatis[10] (1.9.30.689)	database	4,472	1,445	1,284(88.8%)	781(54.1%)
sum	(NONE)	70,602	13,563	12,029(86.8%)	6,678(48.2%)

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．Qualitas Corpus に含まれていない BlueJ，OrderPortal，Saxon-HE，NeoDatis の 4 つのに対して適用した．この 4 つのソフトウェアは，Hayase ら [10] が用いたソフトウェアから選択した．選択した 4 つのソフトウェアについての詳細は表 2 に示す．表 2 においてドメインは，Hayase らが用いたソフトウェア群について行った分類を示す．

結果を表 2 および図 7 に示す．4 つのソフトウェア全体では，13,860 個のメソッドのうち，86.8%のメソッドに対して適切な動詞を提示できた．また，48.2%のメソッドに対しては，上位 5 位に適切な動詞を提示できている．ソフトウェアごとの差異としては，候補リストの上位 5 位までについては 4 つのソフトウェアの中で最もよく推薦できているソフトウェアが NeoDatis であり，最も推薦できなかったソフトウェアが OrderPortal であった．一方，候補リスト内のどこかに提示できているメソッドついては，4 つのソフトウェアの中で最もよく推薦できているソフトウェアが OrderPortal であり，最も推薦できなかったソフトウェアが BlueJ であった．OrderPortal は，上位に正解を提示できているメソッドが少ないものの，リスト内に提示できているメソッドの割合は他のソフトウェアと大きな差がないことがわかる．ここから，ドメインやソフトウェアによって上位に提示できるかどうかに差があると考えられる．これにより，学習に用いていないメソッドに対しても適切な動詞を推薦できているが，正解を提示できる順位がソフトウェアによって一定ではないと考えられる．

[7] http://www.bluej.org/
[8] https://launchpad.net/orderportal
[9] http://saxon.sourceforge.net/
[10] http://sourceforge.net/projects/neodatis-odb/

4.3 評価 2) 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

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．

RQ3: 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

RQ4: 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

4.3.1 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．

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

表 3: 課題として選択したメソッド

課題番号	ソフトウェア	メソッドシグネチャ	順位
1	BlueJ	bluej.graph.SelectionSet.java public void move(int deltaX, int deltaY)	90
2	BlueJ	bluej.views.CallableView.java public void print(FormmattedPrintWriter out, int indents)	5
3	BlueJ	bluej.debugger.jdi.JdiThreadSet.java public JThread find(ThreadReference thread)	5
4	OrderPortal	com.randr.orderportal.util.Utilities.java public static String replace(String where, String replaced, String replacement)	1
5	OrderPortal	com.randr.orderportal.productGroup.ProductGroupController.java private void update()	9
6	OrderPortal	com.randr.scmportal.image.gif.Gif89Encoder.java void put(int rgb, int ipalette)	5
7	Saxon-HE	net.sf.saxon.dom.DocumentBuilderImpl.java public Document parse(InputSource in) throws SAXException	3
8	Saxon-HE	net.sf.saxon.functions.EscapeURI.java public static CharSequence escape(CharSequence s, String allowedPunctuation)	1
9	Saxon-HE	net.sf.saxon.event.Builder.java public void reset()	34
10	NeoDatis	org.neodatis.odb.impl.core.layers.layer3.buffer.MultiBufferedIO.java public void clear()	3
11	NeoDatis	org.neodatis.odb.impl.core.layers.layer3.crypto.AesMd5Cypher.java public void write(byte b) throws IOException	1
12	NeoDatis	org.neodatis.odb.gui.xml.XmlExportPanel2.java private void export() throws Exception	3

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．

節 4.2.2 で用いた 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．選択したメソッドの詳細は表 3 に示す．ソフトウェアは，メソッドが定義されているソフトウェアの名前，メソッドシグネチャは，上段がそのメソッドが定義されているパスを含めた java ファイルの名前，下段は可視性・返り値の型・メソッド名・引数を含めたメソッドシグネチャを表す．順位は，そのメソッドに対して提案手法が正解を提示出来ている順位を表す．

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．

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，ランダムに割り当てた．

アンケート　被験者には課題の解答の後，紙面によるアンケートによって質問に回答してもらった．アンケートでは，被験者の開発経験について，普段のプログラミング時の命名についていくつか質問を行った．普段のプログラミング時の命名について 4 つの質問して 5 段階で回答してもらったほか，名前を付けるときに気を付けていることは何であるかを質問し，自由記述形式で回答してもらった．

質問 1) 命名のときに意味を推測できる名前を付けているか 1. 全くそうしていない，2. そうしていない，3. どちらでもない，4. そうしている，5. とてもそうしている

質問 2) 自分が書いたプログラムを後から読んで理解できることを心がけているか 1. 全く心がけていない，2. 心掛けていない，3. どちらでもない，4. 心掛けている，5. とても心掛けている

質問 3) 自分が書いたプログラムを後から読んで理解できるか 1. 全くそう思わない，2. そう思わない，3. どちらでもない，4. そう思う，5. とてもそう思う

質問 4) 命名のときにどのような命名をしたらいいか悩むことがあるか 1. とてもよく悩む，2. 悩むことがある，3. どちらでもない，4. ほとんど悩んだことはない，5. 全く悩んだことはない

質問 5) 命名のときに何か気を付けていることがあるか 自由記述による回答

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分けて実施を行った．被験者に 1 つの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．

4.3.2 RQ3: 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

提案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ているかについて，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課題ごとに表 4 に示す．表において，提示順位は被験者に提示された候補リスト内に正解の動詞が提示されている順位を表す．総合すると，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．統計検定のための 2 分割表を表 5 に示す．カイ二乗検定における検定力の事前分析を行った．2 の分割表において効果量 0.3，有意水準 0.05，検定力 0.8 であるときの必要標本数が 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．検定力分析には，G*Power3[11] を用い，効果量の設定については [22] を参考にした．

帰無仮説 H_1，対立仮説 H_1' を以下に，計算した結果の p 値を式 (2) に示す．

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と，候補リストがない場合の正解数に差がある．

$$p = 0.54 \tag{2}$$

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．

4.3.3 RQ4: 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．

[11] http://www.gpower.hhu.de/

表 4: 課題ごとの正解数

課題番号	正解の動詞	提示順位	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	move	なし	1	3	1	3
2	print	5 位	3	1	2	2
3	find	5 位	4	0	2	2
4	replace	1 位	4	0	3	1
5	update	なし	1	3	4	0
6	put	5 位	0	4	0	4
7	parse	3 位	2	2	1	3
8	escape	1 位	3	1	3	1
9	reset	なし	0	4	2	2
10	clear	3 位	3	1	1	3
11	write	1 位	4	0	4	0
12	export	3 位	3	1	2	2

正解が提示されている順位ごとの正解数を表 8 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，正解の動詞が本手法で 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 位に正解が提示される

表 5: 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

表 6: 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

ようなメソッドは，62%の人は正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．

また，表 7 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 7 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 6 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' を以下に，計算した結果の p 値を式 (3) に示す.

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と，候補リストがない場合の正解数に差がある.

$$p = 0.045 \tag{3}$$

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．

4.3.4 RQ5: 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

表 7: 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	リスト外の動詞	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

表 8: 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

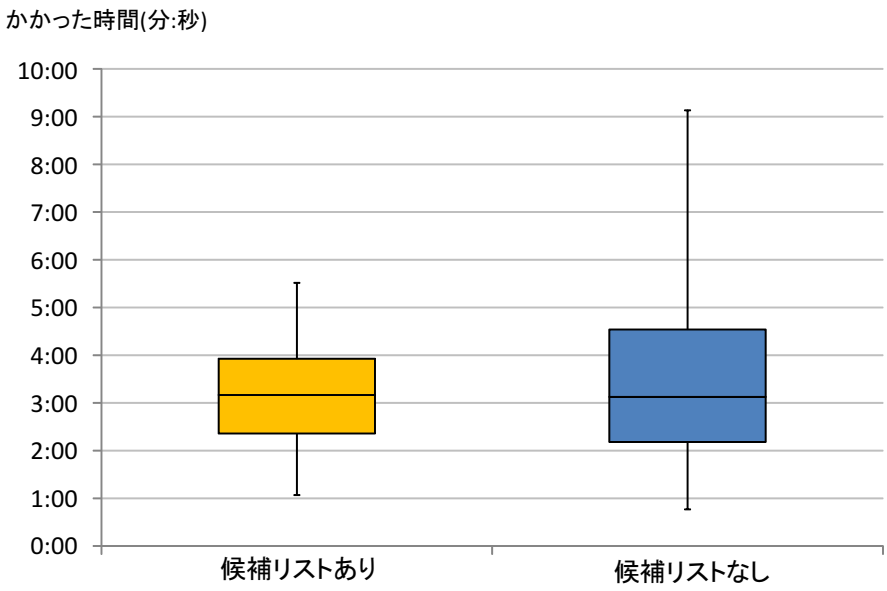

図 8: 候補リストがあるときとないときの課題にかかった時間

表 9: 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

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 8 に，統計的な指標値を表 9 に示す．図 8 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が 3 分 10 秒，候補リストがない場合の平均解答時間が 3 分 34 秒であり，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場合が 3 分 7 秒，ない場合が 3 分 5 秒であり，候補リストの有無による差がなかった．また，候補リストがある方が変動係数が小さいことから，ない場合より解答時間のばらつきが少ないことがわかる．

課題ごとの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 9 に示す．課題ごとに見ると， どの課題についても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．

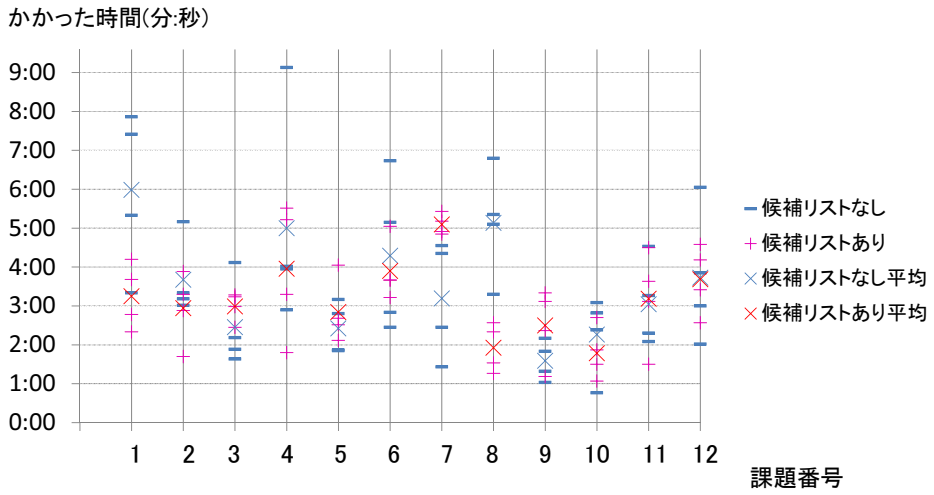

図 9: 課題ごとの候補リストがあるときとないときの課題にかかった時間

4.3.5 考察

結果として，候補リストがある場合の方が正解数は高かったが，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ていることがわかった．

候補リストの有無により，どのようなメソッドについて適切な動詞を選択できていないのか分析を行った．表 4 から，候補リストがない場合の方が正解数の多い課題が 2 つ，正解数が変わらない課題が 4 つ，候補リストがある場合の方が正解数の多い課題が 6 つであり，候補リストがない場合の方が正解数の多い課題は，ともに候補リスト内に正解が提示できていないメソッドであった．

候補リストがない場合の方が正解数が多い課題を図 10，11 に示す．図 10 に示される update メソッドについては，候補リストがある状態で解答した開発者 4 人のうち 2 人が commit メソッド，1 人が refresh メソッドであると解答した．一方で，候補リストがない状態で解答した開発者 4 人は update メソッドであると解答した．commit，refresh ともにメソッドの中で使われている単語であり，候補リストの上位に提示されている動詞であるため，update メソッドではないと判断したと考えられる．このメソッドについては，正解が提示されていなかったことによって動詞の候補リストが開発者の選択を助けなかった例だと考える．

図 11 に示される reset メソッドについては，候補リストがある状態で解答した開発者 4 人のうち 2 人が init メソッド，1 人が initialize，1 人が reset メソッドだと解答した．一方で，

候補リストがない状態で解答した開発者 2 人が init メソッド，2 人が reset メソッドだと解答した．reset メソッドについては候補リスト外の動詞を選択していることが多かった．このメソッドは，すべてがフィールドへの null もしくは bool 値の代入のみである．開発者はこのメソッドがいつ呼び出されるのかがわからず，クラス内の何らかの処理を終わった後にデータを reset するメソッドなのか，処理を始める前に init(initialize) するメソッドなのか判断ができなかったのではないかと考える．このメソッドについては，候補リストによる影響ではなく，課題の実施環境による影響ではないかと考える．

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，以下の 2 つの原因が考えられる．まず，被験者の所属がソフトウェア工学講座であることである．課題の解答後に行ったアンケートの結果を 10 に示す.「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．

次に，出題形式の問題である．被験者は対象となるメソッドおよび，そのメソッド内で使われているフィールド定義，そのメソッドが定義されているクラス情報からそのメソッドの動作を表す動詞を推測した．しかし，課題 (9) のように，メソッドがいつ呼び出されているかによって，同じ動作でも適切な動詞が異なる場合がある．このようなメソッドは，動詞の候補リストの有無にかかわらず適切な動詞を選択することが難しかったと考えられ，被験者に出題する課題として不適切だった可能性がある．

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高かったことから，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，手法が有効に働くのではないかと考える．

```
public class ProductGroupController extends AbstractController {

 private static final String PATH_UPDATE
                        = BASE_PATH + "/updateProductGroup.jsp";

 private void update() {
       try {
            openConnection();
            processUpdate();
            commit();
            refreshMenu();
            forward(PATH_UPDATE);
        } catch (Exception e) {
            doCatch(e);
        } finally {
            finallyMethod();
        }
 }

}
```

動詞の候補

1.) commit 2.)process 3.)refresh 4.)do 5.) execute

図 10: 課題 (4)update

```
public abstract class Builder implements Receiver {

    protected String systemId;
    protected String baseURI;
    protected NodeInfo currentRoot;
    protected boolean lineNumbering = false;
    protected boolean started = false;
    protected boolean timing = false;
    protected boolean open = false;

    public void reset() {
        systemId = null;
        baseURI = null;
        currentRoot = null;
        lineNumbering = false;
        started = false;
        timing = false;
        open = false;
    }
}
```

動詞の候補

1.) start 2.) enter 3.) leave 4.) write 5.) end

図 11: 課題 (9)reset

表 10: 普段の開発における命名についてのアンケート

	1.	2.	3.	4.	5.
質問 1		1			11
質問 2		2	2	8	
質問 3		4	3	5	
質問 4	3	8	1		

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．

品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 [19].

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 [23]．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 [8].

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 [8]，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手法を提案した．提案手法が適切な候補リストを提示できるかどうかを評価し，提案手法を用いて，候補リストが提示されている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，提案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高かったものの，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ていることがわかった．

今後の課題として，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と考える．また，メソッド抽出リファクタリングにおける抽出したメソッドの命名支援に応用できるのではないかと考えている．

謝辞

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，ご多忙の中，大学院での生活の中で，研究の内容だけでなく，私の進路についてもご相談に乗っていただきました．井上先生のもとで多くのことを学べたこと，3 年間の研究生活を送れたことが，とても大きな糧になりました．心から御礼を申し上げます．本当にありがとうございました．

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，多くのご助言を頂きました．特に，研究の中間段階においては，本研究の方向性について多くのご意見を頂き，それらをご参考にさせて頂くことで本研究を遂行することができました．多くのご指導を頂きました松下先生に心から御礼申し上げます．

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，本研究全体を通して，直接のご指導を賜りました．多くのご助言を頂き，本論文を執筆するまで至ることができました．また，本研究についてだけでなく，ソフトウェア工学に関する知識全般，および研究全般に関しての考え方などについて，様々なご教授を頂きました．本論文ならびに研究生活において，常に多くのご意見を賜りました石尾先生に心から御礼申し上げます．本当にありがとうございました．

筑波大学 大学院システム情報工学研究科 早瀬 康裕 助教には，私が研究室配属されてからの 1 年間，研究に関する直接のご指導を賜りました．早瀬先生のご指導なくして，私はここまで成長することはできませんでした．多くのご意見，ご助言を賜りました早瀬先生に心から御礼申し上げます．

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 博士後期課程 鹿島 悠 氏には，本研究において，研究の方針，論文の執筆，資料の作成についてなど多くのご助言を頂きました．研究についてだけでなく，研究室で生活していく上でも，親身になって相談に乗っていただいたこと，心から感謝しています．本当にありがとうございました．

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 肥後 芳樹 助教には，研究室が異なるにも関わらずさまざまな知識をご教授頂き，また，親身になって相談に乗って頂きました．多くのご助言を頂きましたことを心より感謝いたします．

井上研究室の皆様には，3 年間本当にお世話になりました．私が 3 年間，これほど充実した生活をおくり，成長できたのは，井上研究室で皆様とすごせたからだと確信しています．本当にありがとうございました．

参考文献

[1] Dawn Lawrie, C. Morrell, Henry Feild, and David Binkley. What's in a Name? A Study of Identifiers. In *Proc. ICPC*, pp. 3–12, 2006.

[2] Gail C Murphy, Mik Kersten, and Martin P Robillard. The Emergent Structure of Development Tasks. In *Proc. ECOOP*, pp. 33–48, 2005.

[3] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Using IR methods for labeling source code artifacts: Is it worthwhile? In *Proc. ICPC*, pp. 193–202, 2012.

[4] Steve McConnell. *Code Complete, Second Edition*. Microsoft Press, 2004.

[5] Java Code Conventions, 1997.

[6] Einar W. Høst and Bjarte M. Østvold. The Programmer's Lexicon, Volume I: The Verbs. In *Proc. SCAM*, pp. 193–202, 2007.

[7] Einar W. Høst and Bjarte M. Østvold. Debugging Method Names. In *Proc. ECOOP*, pp. 294–317, 2009.

[8] Shusi Yu, Ruichang Zhang, and Jihong Guan. Properly and Automatically Naming Java Methods: A Machine Learning Based Approach. In *Advanced Data Mining and Applications*, Vol. 7713, pp. 235–246. Springer Berlin Heidelberg, 2012.

[9] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. SIGMOD*, pp. 207–216, 1993.

[10] Yasuhiro Hayase, Yu Kashima, Yuki Manabe, and Katsuro Inoue. Building Domain Specific Dictionaries of Verb-Object Relation from Source Code. In *Proc. CSMR*, pp. 93–100, 2011.

[11] Tom Mens and Tom Tourwe. A Survey of Software Refactoring. *TSE*, Vol. 30, No. 2, pp. 126–139, 2004.

[12] Konstantinos Stroggylos and Diomidis Spinellis. Refactoring–Does It Improve Software Quality? In *Proc. WoSQ*, pp. 10–15, 2007.

[13] Opdyke William F. *Refactoring Object-oriented Frameworks*. PhD thesis, 1992.

[14] Gabriele Bavota, Bernardino De Carluccio, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Orazio Strollo. When Does a Refactoring Induce Bugs? An Empirical Study. In *Proc. SCAM*, pp. 104–113, 2012.

[15] Emerson Murphy-hill, Chris Parnin, and Andrew P Black. How We Refactor, and How We Know It. *TSE*, Vol. 38, No. 1, pp. 5–18, 2012.

[16] Opdyke William F. and Johnson Ralph E. Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems. In *SOOPPA*, pp. 145–161, 1990.

[17] Venera Arnaoudova, Laleh Eshkevari, Massimiliano Di Penta, Rocco Oliveto, Giuliano Antoniol, and Yann-Gael Gueheneuc. REPENT: Analyzing the Nature of Identifier Renamings. *TSE*, Vol. 40, No. 5, pp. 502–532, 2014.

[18] Surafel Lemma Abebe, S. Haiduc, P. Tonella, and A. Marcus. Lexicon Bad Smells in Software. In *Proc. WCRE*, pp. 95–99, 2009.

[19] Venera Arnaoudova, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gael Gueheneuc. A New Family of Software Anti-patterns: Linguistic Anti-patterns. In *Proc. CSMR*, pp. 187–196, 2013.

[20] Edvard K. Karlsen, Einar W. Høst, and Bjarte M. Østvold. Finding and fixing Java naming bugs with the Lancelot Eclipse plugin. In *Proc. PEPM*, pp. 35–38, 2012.

[21] Takayuki Suzuki, Kazunori Sakamoto, Fuyuki Ishikawa, and Shinichi Honiden. An Approach for Evaluating and Suggesting Method Names using N-gram Models. In *Proc. ICPC*, pp. 271–274, 2014.

[22] 水本篤, 竹内理. 効果量と検定力分析入門 -統計的検定を正しく使うために-, July 2011.

[23] Robert Green and Henry Ledgard. Coding Guidelines: Finding the Art in the Science. *Communications of the ACM*, Vol. 54, pp. 57–63, 2011.

付録

ルールの抽出に用いた Qualitas Corpus 内のソフトウェア

本研究では，メソッド名の動詞の候補を提示するためのルールを抽出するために，Qualitas Corpus として示されている 112 個のソフトウェアを用いた．Qualitas Corpus のホームページに示されている表をもとに，本研究で用いたソフトウェアについての情報を表 11 に示す．表 11 において，System はソフトウェアの名前，Domain は Qualitas　Corpas の中で定義されているドメインの分類，#Class は，ソフトウェア内部に存在するソースコードとクラスの組の数，#Loc は，ソフトウェアの空白・コメントを除いた行数を表している．

表 11: ルールの抽出に用いた Qualitas Corpus 内のソフトウェア

System	Domain	#Class	#LOC
trove-2.1.0	SDK	34	2196
webmail-0.7.10	tool	104	8212
marauroa-3.8.1	games	204	13823
freecs-1.3.20100406	tool	147	23012
drawswf-1.2.9	3D/graphics/media	319	27008
ganttproject-2.0.9	tool	1058	47051
proguard-4.5.1	tool	658	55567
itext-5.0.3	diagram generator /data visualization	544	76369
findbugs-1.3.9	testing	1715	109096
poi-3.6	tool	1785	143507
megamek-0.35.18	games	2185	258957
netbeans-6.9.1	IDE	31023	1890536
fitjava-1.1	testing	61	2240
jsXe-04_beta	tool	107	8829
jrat-0.6	testing	250	14146
checkstyle-5.1	IDE	349	23316
fitlibraryforfitnesse-20100806	testing	892	27539
jstock-1.0.7c	tool	867	48842
exoportal-v1.0.2	diagram generator /data visualization	1225	56805
c_jdbc-2.0.2	database	586	81306

nakedobjects-4.0.0	IDE	3002	110378
springframework-3.0.5	middleware	3089	160302
jboss-5.1.0	middleware	3612	281643
eclipse_SDK-4.3	IDE	33874	2442717
jFin_DateMath-R1.0.1	SDK	62	4807
picocontainer-2.10.2	middleware	242	9259
joggplayer-1.1.4s	3D/graphics/media	130	14936
axion-1.0-M2	database	261	23744
collections-3.2.1	tool	458	27635
wct-1.5.2	tool	724	49933
pmd-4.2.5	testing	926	60875
sandmark-3.4	tool	1088	90121
aoi-2.8.1	3D/graphics/media	862	111725
jruby-1.5.2	programming language	4664	160360
lucene-4.3.0	tool	3729	285804
quilt-0.6-a-5	testing	77	5683
informa-0.7.0-alpha2	middleware	170	9722
jag-6.1	tool	255	15733
jgraphpad-5.10.0.2	tool	431	23750
sablecc-3.2	parsers/generators/make	285	28394
freemind-0.9.0	diagram generator /data visualization	912	50198
heritrix-1.14.4	tool	703	61681
jmeter-2.9	testing	1143	90612
jrefactory-2.9.19	tool	1553	113427
tomcat-7.0.2	middleware	1739	166478
jtopen-7.1	middleware	2054	397220
jasml-0.10	tool	49	5732
quickserver-1.4.7	middleware	111	10885
xmojo-5.0.0	middleware	135	17669
ivatagroupware-0.11.3	middleware	222	23786
openjms-0.7.7-beta-1	middleware	560	33905
roller-4.0.1	tool	587	50980
drjava-stable-20100913-r5387	IDE	1866	62380

jchempaint-3.0.1	SDK	1045	90831
castor-1.3.1	middleware	1327	115543
jasperreports-3.7.3	diagram generator /data visualization	1844	170064
compiere-330	tool	2534	400257
oscache-2.4.1	middleware	75	6198
displaytag-1.2	diagram generator /data visualization	131	11832
batik-1.7	3D/graphics/media	2599	17848
emma-2.0.5312	testing	330	25806
jung-2.0.1	diagram generator /data visualization	858	37989
mvnforum-1.2.2-ga	tool	273	51034
jena-2.6.3	middleware	1392	70948
jgroups-2.10.0	tool	1211	96325
myfaces_core-2.0.2	middleware	1365	119529
xalan-2.7.1	parsers/generators/make	1238	189462
aspectj-1.6.9	programming language	2624	412394
nekohtml-1.9.14	parsers/generators/make	55	6625
jgrapht-0.8.1	tool	255	11931
log4j-1.2.16	testing	308	20637
jext-5.0	diagram generator /data visualization	504	26565
colt-1.2.0	SDK	593	38625
cobertura-1.9.4.1	testing	122	51860
columba-1.0	tool	1190	71680
jfreechart-1.0.13	tool	649	98078
hsqldb-2.0.0	database	535	123268
argouml-0.34	diagram generator /data visualization	2560	192410
gt2-2.7-M3	SDK	5593	446863
squirrel_sql-3.1.2	database	169	6944
jparse-0.96	parsers/generators/make	69	12559
sunflow-0.07.2	3D/graphics/media	221	21648
quartz-1.8.3	middleware	286	26819
htmlunit-2.8	testing	556	40004

galleon-2.3.0	3D/graphics/media	790	52653
rssowl-2.0.5	tool	1682	73230
freecol-0.10.7	games	1310	100748
cayenne-3.0.1	database	2171	127529
hibernate-4.2.2	database	3614	217163
azureus-4.8.1.2	database	7680	484739
junit-4.11	testing	221	7428
jpf-1.5.1	SDK	189	13246
antlr-4.0	parsers/generators/make	385	21919
velocity-1.6.4	diagram generator /data visualization	261	26854
jspwiki-2.8.4	middleware	455	43326
tapestry-5.1.0.5	middleware	1502	53367
struts-2.2.1	middleware	1074	74670
ant-1.8.4	parsers/generators/make	1290	105007
xerces-2.10.0	parsers/generators/make	948	129164
ireport-3.7.5	diagram generator /data visualization	3381	221490
derby-10.6.1.0	database	2995	592817
jmoney-0.4.4	tool	193	8197
javacc-5.0	parsers/generators/make	100	13772
jgraph-5.13.0.0	tool	187	22758
james-2.2.0	tool	340	27003
pooka-3.0-080505	tool	849	44474
maven-3.0	parsers/generators/make	697	54336
jhotdraw-7.5.1	3D/graphics/media	1070	75958
jedit-4.3.2	tool	1128	107469
hadoop-1.0.0	middleware	2069	142790
weka-3.7.9	tool	2390	247805
jre-1.6.0	programming language	10714	922958