Vol.2016-SE-191 No.17
2016/3/14

クラス名命名を支援する段階的なフレーズ推薦手法

福田 宏樹[1,a)]　早瀬 康裕[1,b)]　北川 博之[1,c)]

概要：ソフトウェアの保守作業において，クラス名などの識別子がプログラム理解の際の重要な手がかりになることが知られている．しかし，役割や意味が分かりやすいクラス名を付けることは，専門知識を必要とする困難な作業である．そこで本研究では，クラス名の改名のような，既に内容が存在するクラスに対して名前をつける場面を対象として，開発者とシステムとが対話的にクラス名を組み立てていく手法を提案する．システムはクラス名に相応わしいと考えられるフレーズのリストを推薦し，開発者はフレーズを 1 つ選択する．続いて，システムは選択されたフレーズに付け加えるフレーズのリストを推薦する．フレーズの推薦には，既存のソースコードから学習した，クラス名と内容の関連を利用する．

1. はじめに

近年のソフトウェア開発では，ソフトウェアの大規模化，複雑化により，ソフトウェア保守にかかるコストの増加が問題となっている．ソフトウェア保守にかかるコストはソフトウェアのライフサイクル全体でかかるコストのおよそ 3 分の 2 を占めると言われており [17]，このコストを削減することは大きな課題である．

ソフトウェア保守のために保守作業者が行う作業の中でも，対象のプログラムの動作を理解するという作業にかかるコストは大きいとされる．なぜならば，ソフトウェア保守を行うためには，まず，プログラム中の修正すべき箇所がどこであり，どのように直せば修正ができるかを，対象となるプログラムから読み取る必要があるためである．このプログラム理解にかかる時間は，ソフトウェア保守にかかる時間の中でも半分以上を占めるとされる [4]．

保守作業者が対象のプログラムを理解する際，プログラムのソースコード中に出現する，クラス名やメソッド名などの識別子名を大きな手がかりの一つとして用いることが知られている．なぜならば，保守作業者は，識別子名の意味からそのプログラム要素の役割を推測し [16]，プログラム理解に役立てるためである．そのため，開発者があらかじめ識別子に理解しやすい名前を付けておくことで，保守作業者が対象のプログラムを理解するのにかかる時間を減らすことができる [12]．

しかしながら，識別子に理解しやすい名前を付けることは，一般に専門知識を必要とする難しい作業である．専用の教材や命名規約は処理と名前の対応関係についてわずかなガイドラインを提供するのみであるので，適切な命名に必要な知識は，主に実際の開発で専門的な経験を積んでいくことでしか得られない．

開発者は必ずしも全てのドメインに通じているわけではないので，命名に必要な専門知識を全てのドメインについて持っているとは限らない．そのため，どのような開発者であっても，特に経験の浅いドメインの開発に取り組む場合には，識別子への適切な命名が困難となることがある．

開発者にとって負担となる作業は統合開発環境による補助が行われることが多いが，現在の統合開発環境が持つ識別子への命名を補助する機能はごく単純なものに限られており，十分なものとは言えない．例えば，Java の統合開発環境の一つである Eclipse[*1] は，getter や setter のような名前が機械的に決まるメソッドの命名は自動で行えるが，それを超える高度な推薦は行っていない．

そこで，識別子の命名を支援することを目的として，識別子名を推薦する研究がこれまでにいくつか行われている．Høst ら [9] はメソッドの実装上の特徴に基づいてメソッド名の推薦を行い，Allamanis ら [2] は実装上の特徴と周辺のトークンの情報に基づいて，変数名，メソッド名，クラス名を推薦する研究を行った．これらの研究に共通する点として，1〜5 個程度の少数の候補を高い確信に基づいて推薦するアプローチをとっているという点が挙げられる．このアプローチは，開発者にとって信頼できる推薦結果が得られるという利点がある．

1 筑波大学
University of Tsukuba
a) hfukuda@kde.cs.tsukuba.ac.jp
b) hayase@cs.tsukuba.ac.jp
c) kitagawa@cs.tsukuba.ac.jp

*1 https://eclipse.org/

© 2016 Information Processing Society of Japan

しかしながら，これらの研究がとっているアプローチにはある点で限界がある．注目する点は，識別子名は唯一の正解があるわけではないものであるという点である．識別子名というものはその識別子が表すメソッドやクラスなどのプログラム要素の性質を簡潔に表現していればよく，そのプログラム要素のどの性質をどのように表現するかに応じて，複数の名前が適切な識別子名となりうる．そのため，理想的には，推薦システムは複数ありうる適切な識別子名を候補として推薦するのみにとどめ，その中から開発者が自らの判断基準に応じて最も相応しいと感じた識別子名を選べるようになっていると望ましい．しかし，既存の研究は少数の候補しか推薦しないため，開発者がより相応しいと考える識別子名を見逃してしまう可能性がある．

このような推薦に近づくためのアプローチの一つとして考えられるものは，開発者がより相応しいと考える識別子名を得るためのヒントとして，多数の候補を開発者に推薦するアプローチである．多数の候補を推薦すれば開発者がより相応しいと考える識別子名を見逃してしまう可能性は少なくなる．

しかしながら，多数の候補を推薦するアプローチには問題がある．開発者が一度に検討しなければならない候補の数が増えてしまい，開発者の負担が増してしまうという問題がある．

そこで，本研究では，理解しやすい識別子名を命名することが開発者にとって難しいという問題を軽減するために，識別子名をフレーズ単位で段階的に推薦していく手法を提案する．識別子名は複数の単語が組み合わさってできているが，フレーズとはその識別子名の部分単語列を指す言葉である．推薦システムはクラス名に相応しいと考えられるフレーズのリストを推薦し，開発者はフレーズのリストを一つ選択する．続いて，システムは選択されたフレーズに付け加えるフレーズのリストを推薦する．このように，フレーズという細かい構成要素ごとの単位で推薦を行っていき，対話的に識別子名を組み立てていくことで，開発者が一度に検討しなければならない候補の数を減らしつつ，最終的に得ることができる可能性がある識別子名の数を維持することができる．

本研究では識別子名の中でもまずクラス名に注目し，クラス名の推薦を行う．なぜならば，クラス名は，識別子名の中でもプログラムの読みやすさにより直結するので，より重要性が高いからである．例えば，クラス名からクラスの働きが推測できれば，プログラム全体というマクロなレベルの理解にも，そのクラス内部の処理というミクロなレベルの理解にも役に立つ．

本研究では，クラス名の中でも，特に中心的な形である，名詞句の形をとるクラス名を推薦できるようにする．Butler ら [3] はクラス名の多数が名詞句を占めると報告しており，著者ら [6] も同様の事実を確認している．そのような多数を占める形のクラス名をまず推薦できるようにすることで，適切なクラス名のうちの多くをカバーできることが期待される．

本研究で対象とする状況は，クラスの内容が既に記述済みであるときにクラスに名前をつける状況である．そのような状況には，クラスの改名や，Extract Class[5] のような，リファクタリングの結果として新しいクラスが現れるリファクタリングがありうる．

フレーズの推薦は，クラスの内容に基づいて行う．そのために，既存のソースコードから学習した，クラス宣言内の識別子とクラス名のフレーズとの関連を利用する．柏原ら [11] は，メソッド周辺の識別子とメソッド名の動詞部分とに強い関連があることを確認しており，クラスにおいても同様の関係が成り立つことが期待されると考えたので，推薦材料としてクラス宣言内の識別子を用いる．

2. 前提知識

本研究ではクラスの内容とクラス名のフレーズとの関係を相関ルールマイニング [1] を用いて抽出する．相関ルールマイニングについて 2.1 節で，クラス名について 2.2 節で説明する．

2.1 相関ルールマイニング

相関ルールマイニングとは，ある事象の組み合わせが同時に発生したとき，別のある事象の組み合わせも同時に発生することが多いという関係（相関ルール）を抽出する手法の一つである．相関ルールマイニングでは，一つの事象をアイテムと呼び，同時に発生した複数の事象の組み合わせをトランザクションと呼ぶ．トランザクションはアイテムの集合として表現される．相関ルールマイニングは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，相関ルールの集合を抽出する．

ある事象の組み合わせ X が同時に発生したとき，別のある事象の組み合わせ Y も同時に発生することが多いという関係を表す相関ルールは，$X \rightarrow Y$ のように表記する．X は相関ルールの条件に当たる部分であり，条件部と呼ばれる．Y は相関ルールの結論に当たる部分であり，帰結部と呼ばれる．X と Y は互いに排反でしかも空でないアイテム集合になる．

相関ルールが有用なものかどうかを評価する代表的な指標として，確信度と支持度がある．確信度とは，X が現れたときに Y も同時に現れる確率のことである．支持度とは，そもそも $X \rightarrow Y$ という関係が成り立っているトランザクションの数のことである．相関ルールマイニングでは有用な相関ルールを抽出するために，確信度と支持度がそれぞれ一定の閾値以上のルールを抽出する．この閾値は相関ルールマイニングを行う人間が目的に応じて手動で設定する．

© 2016 Information Processing Society of Japan

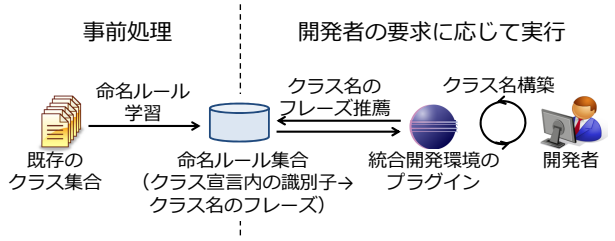

図 1　提案手法の概要図

2.2　クラス名

クラス名は，クラスの役割を表現するために，一般に，1 単語以上の複合語を用いて命名する．多くのプログラミング言語ではクラス名の中にスペースを入れることはできないため，単語同士をスペースを用いずに連結する記法として，主に以下の 4 つの記法が用いられている．

`camelCase`　1 単語目の頭は小文字，2 単語目以降の頭は大文字にして単語を連結する記法．

`PascalCase`　各単語の頭を大文字にして単語を連結する記法．

`snake_case`　各単語を全て小文字にし，スペースの代わりにアンダースコアで単語間を連結する記法．

`SCREAMING_SNAKE_CASE`　各単語を全て大文字にし，スペースの代わりにアンダースコアで単語間を連結する記法．

Java の命名規約 [13] を含む多くの命名規約ではクラス名に名詞句を用いることが推奨されており，実際に多くのクラス名は名詞句であることが確認されている．Butler ら [3] は，85%のクラス名は 1 単語以上の連続する名詞の左に 0 単語以上の連続する形容詞を持つ形であると報告している．著者ら [6] も同様に，名詞句がクラス名の多数を占めることを確認している．

3.　提案手法

本研究では，理解しやすいクラス名を命名するのが開発者にとって難しいという問題を軽減するために，クラス名の改名のような，既に内容が存在するクラスに対して名前をつける場面を対象として，開発者とシステムとが対話的にクラス名を組み立てていく手法を提案する．本研究では数あるクラス名のうち，多数を占めると報告されている [3], [6]，名詞句の形をとるクラス名に注目する．名詞句の形をとるクラス名は，最も右に核となる名詞句があり，その左にクラスの性質を限定する名詞句もしくは修飾語句がいくつか付く形をとっている．そのような形のクラス名を作ることができる手法を提案する．提案手法を用いる開発者は，まずクラス名の核となる名詞句を推薦されたリストの中から選んだ後，そのフレーズの左に付け加える名詞句または修飾語句を好きなだけ推薦されたリストの中から選んでいくことで，クラス名を組み立てていく．

提案手法では推薦に利用する情報源として，既存のソースコードから抽出した，クラスの内容とクラス名のフレーズとの関係を表すルールを用いる．このルールを本稿では命名ルールと呼ぶ．命名ルールは具体的には，クラス宣言内の識別子の組み合わせと，クラス名のフレーズとの関係を表す相関ルールである．

提案手法の概要を図 1 に示す．図 1 のように，提案手法は，命名ルールを学習しておく事前処理の部分と，事前処理で学習した命名ルールを活用し，開発者の要求に応じてクラス名を組み立てていく処理の部分とに分かれている．後者の，開発者の要求に応じて実行する処理は，統合開発環境に組み込まれたプラグインとして実装される．このプラグインは，さらに，開発者に対話的な UI を提供してクラス名を構築していくフロントエンド部分と，フロントエンドから発行されたクエリに応じてクラス名のフレーズ推薦を行うバックエンド部分とに分かれている．

提案手法を構成する 3 つの処理について各節で詳しく説明する．命名ルール学習の処理については 3.1 節で，クラス名のフレーズ推薦の処理については 3.2 節で，クラス名構築の処理については 3.3 節で説明する．

3.1　命名ルール学習

命名ルール学習は，既存の大量のソースコードからクラスの内容とクラス名のフレーズとの関連を学習する処理である．入力は，既存のソースコード中に記述されているクラスの集合であり，出力は命名ルールの集合である．

命名ルールは特定の形を持つ相関ルールとして表される．すなわち，条件部がクラス宣言内の識別子を表すアイテム複数から成っていて，帰結部がクラス名のフレーズを表すアイテム 1 個から成っている相関ルールを命名ルールとして扱う．

命名ルール学習は 3 ステップに分けて処理を行う．すなわち，既存のクラス集合からトランザクション集合を作り，その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，その結果から命名ルールとして扱える特定の形を持った相関ルールのみを選択する．各ステップについてそれぞれ 3.1.1 節，3.1.2 節，3.1.3 節で説明する．

3.1.1　クラスをトランザクションの形にする

このステップでは，1 つのクラスを 1 つのトランザクションにすることで，相関ルールマイニングの対象とするトランザクション集合を作る．入力は，既存のソースコード中に記述されているクラスの集合であり，出力はトランザクションの集合である．

トランザクションを作るためにクラス内の各要素をアイテムに変換する．アイテムには大きく分けて 2 種類があ

り，一つはクラス宣言内の識別子を表すアイテム，もう一つはクラス名のフレーズを表すアイテムである．クラス宣言内の識別子を表すアイテムは，クラス宣言内にある親クラス名，親インタフェース名，フィールドの型名，フィールド名，メソッドの返り値の型名，メソッド名をアイテムに変換したものである．これらのアイテムは，識別子の種類:識別子の名前，という形でアイテムになる．クラス名のフレーズを表すアイテムは，クラス名が持つ部分単語列を全てアイテムに変換したものである．これらのアイテムは，`class-name-phrase`:クラス名の部分単語列，という形でアイテムになる．

あるクラスがトランザクションとしてどのように表現されるのかの例を図 2 に示す．図 2(a) は `LinkedNameList` という名前のクラスを記述したソースコードであり，図 2(b) は `LinkedNameList` クラスに対応するトランザクションである．

部分単語列を得る際はクラス名を単語分割することになる．単語分割の際は，クラス名が `camelCase`，`PascalCase`，`snake_case`，`SCREAMING_SNAKE_CASE` のいずれかであると仮定し，アンダースコアまたは大文字小文字の境界で単語を分割する．アンダースコアと大文字小文字が混じっており，明らかに上記の 4 方式のいずれとも見なせないクラス名を持つクラスはこの段階で知識の抽出対象から外す．

3.1.2 相関ルールマイニング

このステップでは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．入力は 3.1.1 節で得たトランザクション集合であり，出力は相関ルールの集合である．

相関ルールマイニングの閾値は多くの関係を抽出できるように，計算時間の許す限り低い値にする．すなわち，確信度の閾値については 0 とし，支持度の閾値については出来る限り低くする．

3.1.3 命名ルールの選択

このステップでは相関ルールの中から命名ルールとして解釈できる特定の形の相関ルールを選択する．入力は 3.1.2 節で得た相関ルールの集合であり，出力は命名ルールの集合である．

命名ルールとして，次の 2 条件を共に満たす相関ルールのみを選択する．

- 条件部が，クラス宣言内の識別子を表すアイテムのみから成る
- 帰結部が，クラス名のフレーズを表すアイテム 1 個のみから成る

3.2 クラス名のフレーズ推薦

クラス名のフレーズ推薦は，UI から受けたリクエストに応じて実際に推薦を行う処理である．入力は，開発者が推薦を要求しているクラスの内容と，3.1 節で得た命名ルールの集合であり，出力はクラス名のフレーズのリストであ

```
public class LinkedNameList
        implements Serializable {
    LinkedList<String> nameList;
    int size;
    public String findName(String n) {
        if (nameList.contains(n)) {
            return n;
        }
        return null;
    }
}
```

(a) クラスの例

```
{
 super-interface-name:Serializable,
 field-type-name:LinkedList,
 field-name:nameList,
 field-type-name:int,
 field-name:size,
 return-value-type-name:String,
 method-name:findName,
 class-name-phrase:Linked,
 class-name-phrase:Name,
 class-name-phrase:List,
 class-name-phrase:LinkedName,
 class-name-phrase:NameList,
 class-name-phrase:LinkedNameList
}
```

(b) (a) のクラスに対応するトランザクション

図 2 トランザクションの例

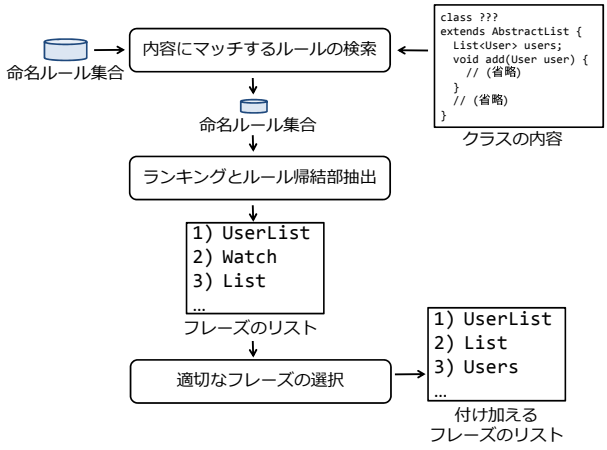

図 3 推薦エンジン

る．このとき出力するフレーズは，1 段階目の推薦ではクラス名の核となる名詞句であり，2 段階目以降の推薦では既に存在するフレーズの左に付け加える名詞句または修飾語句となる．この処理を行う推薦エンジンの概要を図 3 に示す．図 3 に沿って各ステップについて説明する．

3.2.1 内容にマッチするルールの検索

内容にマッチするルールの検索では，ルール学習の処理で事前に抽出した命名ルール集合の中から，開発者が指定したクラスの内容と関連の強い命名ルールを検索する．入力は，開発者が指定したクラスの内容と，3.1 節で事前に抽出した命名ルールの集合であり，出力は，開発者が指定したクラスの内容と関連の強いルールの集合である．

検索の結果として得るルールは，命名ルールの条件部が，開発者が指定したクラスから得たクラス宣言内の識別子の部分集合となっているものである．すなわち，R を事前に抽出した命名ルールの集合，A_r を命名ルール r の条件部，Q を開発者が指定したクラスから得たクラス宣言内の識別子の集合としたとき，以下の条件を満たす命名ルール r を検索する．

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

命名ルールの検索には鬼塚らによる部分集合検索アルゴリズム [15] を用いた．このアルゴリズムは大きな相関ルール集合に対して高速に上記のルール検索を行える．

3.2.2 ランキングとルール帰結部抽出

ランキングとルール帰結部抽出では，検索で得られた命名ルールを推薦度順にランキングし，帰結部を抽出してフレーズのリストとする．入力は，3.2.1 節で得た命名ルールの集合であり，出力は，フレーズのリストである．

推薦度として用いるのは，命名ルールの確信度である．確信度を用いる理由は，確信度が，クラスがある識別子の組み合わせを持つときにクラス名のフレーズを持つ確率であると見なせるからである．

命名ルールの帰結部を抽出してフレーズのリストを作る．この際，同じフレーズを持つ命名ルールが複数現れることがあるので，推薦度が最高の命名ルールを残して排除する．

3.2.3 適切なフレーズの選択

適切なフレーズの選択は，得たフレーズのリストから現在の状況に適切なフレーズを選択する処理である．入力は，3.2.2 節で得たフレーズのリストであり，出力は，1 段階目の推薦では名詞句のリスト，2 段階目以降の推薦では名詞句または修飾語句のリストである．

フレーズを選択するためにフレーズが名詞句や修飾語句であるかどうか，自作の候補品詞判定器 [6] を用いて判定する．この候補品詞判定器は，ある単語が持っている可能性のある品詞を 1 個以上出力する．候補品詞判定器の実装は，WordNet[14] をベースにいくつかのヒューリスティクスを追加したものである．候補品詞判定器は品詞の判定に失敗したとき，それを名詞単数形であると見なすので，必ず何かの候補品詞を出力する．

候補品詞判定器を用いて 1 段階目の推薦では名詞句と判定されたフレーズを，2 段階目以降の推薦では名詞句または修飾語句と判定されたフレーズのみを推薦する．名詞句と判定されるフレーズは，最後の単語が名詞単数形か名詞複数形のどちらか 1 つ以上を候補品詞として持っており，かつ，その他の単語が全て，名詞単数形か名詞複数形か動詞現在分子か動詞過去分詞か形容詞のどれか 1 つ以上を候補品詞として持つフレーズである．修飾語句と判定されるフレーズは，全ての単語が形容詞か動詞現在分詞か動詞過去分詞のどちらか 1 つ以上を候補品詞として持っているフレーズである．

3.3 クラス名構築

提案手法を実装したシステムを利用する開発者は対話的にクラスの内容に合ったクラス名を組み立てていく．この開発者とシステムとの間のインタラクションは図 4 のようにして行われる．

図 4 にしたがって説明を行う．まず，開発者がシステムに対してクラス名のフレーズの推薦を要求する．すると，システムはクラスの内容に基づいて，クラス名の核となる名詞句のリストを推薦する．開発者は推薦されたリストの中から一つだけ好きなフレーズのリストを選ぶ．開発者は選んだフレーズの一部を修正したければ修正することができる．この時点で開発者が現在のフレーズを新しいクラス名として使いたいと考えれば，推薦を完了し，クラス名を現在のフレーズで置き換えることができる．開発者が現在のフレーズに不足を感じており付け足すフレーズを欲しいと考えれば，開発者はさらに左に付け足す名詞句または修飾語句の推薦を要求することができる．システムは，クラ

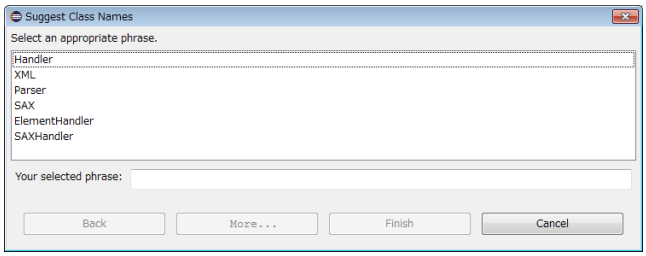

図 5　提案システムが開発者に提示するダイアログ

スの内容に基づいて，クラス名の性質を限定する名詞句または修飾語句のリストを推薦する．以降は繰り返しとなる．

既にシステムのプロトタイプを Eclipse のプラグインとして作成している．このプロトタイプでは，一連の推薦を図 5 のようなモーダルダイアログの中で行う．クラス名のフレーズのリストは Recommended phrases 欄に表示され，開発者は好きなフレーズを選択することができる．選択したフレーズは Your selected phrase 欄に表示され，自由に編集することができる．More... ボタンで追加するフレーズの推薦を要求することができ，Finish ボタンで推薦を完了することができる．また，Back ボタンで推薦の段階を一段階前に戻すことができ，Cancel ボタンで一連の推薦をキャンセルすることができる．

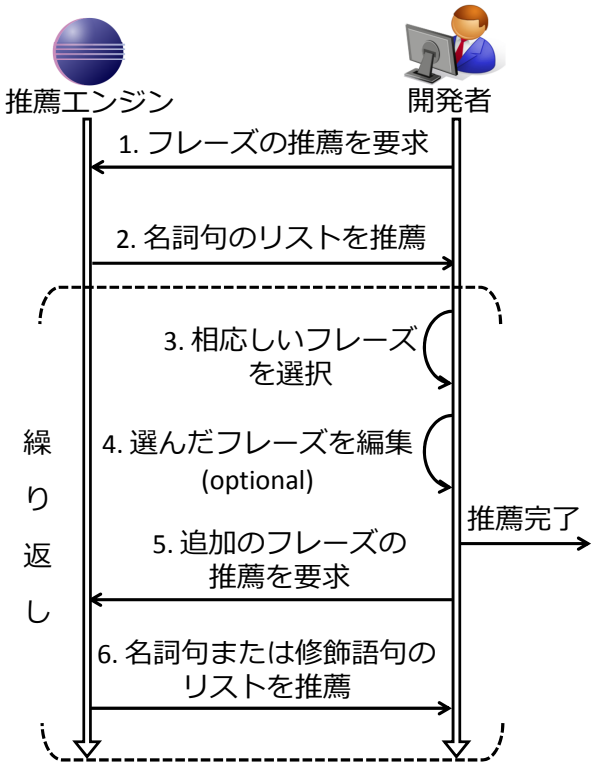

図 4　クラス名構築の流れ

4.　実験計画

提案手法を用いることで，開発者がクラスに適切な命名をする際の負担を軽減できるかどうか評価するために，被験者実験を行うことを計画している．被験者実験では評価できるデータの規模に限界があるため，コンピュータ上で疑似的な推薦のシミュレーション実験を行うことも計画している．被験者実験について 4.1 節で，コンピュータ上でのシミュレーション実験について 4.2 節で説明する．

4.1　被験者実験

提案手法を用いることで，開発者がクラスに適切な命名をする際の負担を軽減できるかどうか評価するために，被験者実験を行うことを計画している．計画している評価項目は，具体的には以下の 3 点である．

- 開発者がクラスに名前をつけるのにかかる負担は軽減できたか
- 保守作業者がクラスの働きを正確に理解できるようになったか
- 保守作業者がクラスの働きを理解するのにかかる負担は軽減できたか

それぞれの項目を評価するために，提案手法を用いた場合と用いなかった場合とで，以下の指標がどう変化するか測定することを計画している．

- 開発者役の被験者がクラス名をつけるのにかかった時間

- 開発者役の被験者がつけたクラス名を持つクラスに対して，保守作業者役の被験者がクラスの働きを正答できたクラスの割合
- 開発者役の被験者がつけたクラス名を持つクラスに対して，保守作業者役の被験者がクラスの働きを回答するのにかかった時間

各指標を評価するために，被験者を十名ほど用意して，開発者役と保守作業者役とに分けることを予定している．開発者役の被験者には，クラス名がつけられていないクラスにクラス名をつけるタスクを行ってもらい，保守作業者役の被験者には，開発者役の被験者がつけたクラス名を持つクラスを与えて，そのクラスがどのようなクラスなのか回答するタスクを行ってもらう．

4.2 コンピュータ上での擬似的な推薦の実験

被験者実験は規模を大きくすることが難しいため，データ量を確保して評価の妥当性を高めるために，コンピュータ上で疑似的な推薦を行う実験を計画している．この実験では，保守作業者がクラスの働きを正確に理解できるようなクラス名を推薦する能力が，提案手法にあるかどうかを評価する．そのために，元のクラス名が正しいものと仮定し，クラスの内容だけから元のクラス名を推薦し直せるかどうかを評価する．データセットとして，OSS ホスティングサイトから既存の大量のソースコードを用意し，それらに対して交差検定を行うことで推薦能力の評価を行う．

使用するデータセットの用意は既に行っている．このデータセットは，OSS ホスティングサイトの一つである SourceForge.net[*2] から Java のプロジェクト 19,329 個を収集したものである．テストコードやバージョン違いのコードは可能な限り除去してある．得たクラスの数は 2,240,715 個である．

5. 関連研究

本研究は，識別子の命名が難しいという問題を軽減するために，ソースコードから命名の知識を抽出した辞書を構築し，その辞書を用いて識別子名を推薦するという二段階の構造を持つ．識別子の命名に用いる知識を抽出または調査する研究について 5.1 節で紹介し，識別子名を推薦する研究について 5.2 節で紹介する．

5.1 識別子の命名に用いる知識を抽出または調査する研究

Høst らは，メソッド名が多くの場合動詞である単語から始まることに注目し，メソッド名の動詞部分を推薦するために，メソッドの実装とメソッド名に用いられている動詞との関係を収録した辞書を構築する手法を提案した [8]．この辞書は，ある動詞をメソッド名に持つメソッドが，どのような実装上の特徴をどれくらいの確率で持つかを収録しており，例えば `get` という動詞をメソッド名に持つメソッドは，多くの場合状態の読み込みを行い，引数を持つことは稀であるという情報を持っている．Høst らはこのような関係性を収録する辞書を既存の Java のソースコード群の情報に基づいて構築する手法を提案し，実際に辞書を作成した．

早瀬らは，ソースコード中に現れる識別子に基づいて，関連の強い動詞-直接目的語-間接目的語のドメイン固有の関係を抽出する手法を提案した [7]．本研究の，クラス宣言内の識別子を推薦に利用するアイデアはこの研究に基づいている．

Butler らは，プログラム理解の支援に活用できる基礎データを得るために，クラス名を構成する単語列がどのような品詞の並びになっていることが多いのか，クラス名用に再学習した品詞判定器を用いて調査を行った [3]．調査の結果として，85%のクラス名は 1 単語以上の連続する名詞の左に 0 単語以上の連続する形容詞を持つ形であるということ，動詞句をはじめとした多様な形のクラス名がロングテール状に分布していること，親クラス名や親インタフェース名の一部がクラス名の一部として継承されることがあることなどが報告されている．本研究の，クラス名の名詞句に注目するというアイデアはこの研究の調査結果に基づいている．

5.2 識別子名を推薦する研究

メソッド名を対象とする研究がいくつか行われている．Høst らは，[8] に基づいて，あるメソッドの実装とメソッド名の動詞とが辞書に収録されている一般的な関係性からずれている場合に，現在のメソッドの実装に基づいて適切な動詞を理由とともに提示する手法を提案した [9]．Høst らはさらに，この手法を実装したツールを Eclipse のプラグインとして作成している [10]．柏原らは，メソッド名の改名時を対象として，開発者が記述したメソッドの周辺に現れる識別子を元に，適切なメソッド名の動詞を推薦度順にリストとして推薦する手法を提案している [11]．これらの研究はメソッド名の動詞部分を対象としているが，本研究はクラス名を対象としている．

Allamanis らは，実装と周辺のトークンとを元に，適切な変数名，メソッド名，クラス名を推薦度順にリストとして推薦する手法を提案している [2]．Allamanis らの研究はクラス名を一度に推薦しているが，本研究はクラス名をフレーズ単位で対話的に組み立てていく．

6. まとめ

本研究ではクラス名の命名が難しいという問題を解決するために，内容のあるクラスへの命名時を対象として，開発者が指定したクラスに対してフレーズの推薦を対話的に

*2 https://sourceforge.net/

行い，クラス名を組み立てていく手法を提案した．クラス名のうち，多数を占めると報告されている名詞句に注目し，そのような形のクラス名を推薦することができるようにした．推薦に用いる情報源として，既存のソースコードから抽出したクラス宣言内の識別子とクラス名のフレーズとの関係を用いた．

クラス名は名詞句以外にも，動詞句など様々な形をとりうることが報告されている．そのような形のクラス名を構築するためには，フレーズの知識を抽出する際の知識を増やせばよいのではないかと考えている．つまり，名詞句由来のフレーズなのか動詞句由来のフレーズなのかを知識を抽出する際記憶しておき，その知識を推薦の際に活用すればよいのではないかと考えている．

SourceForge.net の 19,329 の Java プロジェクトを収集し，テストコードやバージョン違いのコードなどを排除したクラス 2,240,715 個を得た．今後はこのクラスに対して命名ルールの抽出を行い，そのルールを用いて，開発者がクラスに適切な命名をする際の負担を軽減できるかどうか評価実験を行う予定である．

謝辞　本研究は，若手 (B) 25730036 による．

参考文献

[1] Agrawal, R., Imielinski, T. and Swami, A.: Mining association rules between sets of items in large databases, *Proc. of the 1993 ACM SIGMOD Int'l Conf. on Management of Data* (1993).

[2] 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*, ESEC/FSE 2015, New York, NY, USA, ACM, pp. 38–49 (2015).

[3] Butler, S., Wermelinger, M., Yu, Y. and Sharp, H.: Mining Java class naming conventions, *Proc. of the Int'l Conf. on Software Maintenance* (2011).

[4] Corbi, T. A.: Program Understanding: Challenge for the 1990's, *IBM Systems Journal*, Vol. 28, No. 2, pp. 294–306 (1989).

[5] Fowler, M.: Catalog of Refactorings, (online), available from ⟨http://www.refactoring.com/catalog/⟩ (accessed 2016-02-17).

[6] 福田宏樹，早瀬康裕，北川博之：クラス名の単語列に対する品詞列ごとのクラス名数の定量的調査，第 13 回情報科学技術フォーラム (2014).

[7] Hayase, Y., Kashima, Y., Manabe, Y. and Inoue, K.: Building Domain Specific Dictionaries of Verb-Object Relation from Source Code, *Proceedings of the 15th European Conference on Software Maintenance and Reengineering,*, pp. 93–100 (2011).

[8] Høst, E. W. and Østvold, B. M.: The Programmer's Lexicon, Volume I: The Verbs, *Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation*, SCAM '07, Washington, DC, USA, IEEE Computer Society, pp. 193–202 (2007).

[9] Høst, E. W. and Østvold, B. M.: Debugging Method Names, *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, Berlin, Heidelberg, Springer-Verlag, pp. 294–317 (2009).

[10] Karlsen, E. K., Høst, E. W. and Østvold, B. M.: Finding and Fixing Java Naming Bugs with the Lancelot Eclipse Plugin, *Proceedings of the ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation*, PEPM '12, New York, NY, USA, ACM, pp. 35–38 (2012).

[11] Kashiwabara, Y., Onizuka, Y., Ishio, T., Hayase, Y., Yamamoto, T. and Inoue, K.: Recommending verbs for rename method using association rule mining, *Proc. of the IEEE CSMR-WCRE 2014 Software Evolution Week* (2014).

[12] Lawrie, D., Morrell, C., Feild, H. and Binkley, D.: What's in a Name? A Study of Identifiers, *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, Washington, DC, USA, IEEE Computer Society, pp. 3–12 (2006).

[13] Microsystems, S.: Code Conventions for the Java Programming Language, (online), available from ⟨http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html⟩ (accessed 2016-02-17).

[14] Miller, G. A.: WordNet: a lexical database for English, *Communications of the ACM*, Vol. 38, No. 11, pp. 39–41 (1995).

[15] 鬼塚勇弥：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，修士論文，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 (2014).

[16] Pennington, N.: Comprehension strategies in programming, *Proc. of Workshop on Empirical Studies of Programmers* (1987).

[17] Zelkowitz, M. V., Shaw, A. C. and Gannon, J. D.: *Principles of Software Engineering and Design*, Prentice Hall Professional Technical Reference (1979).

© 2016 Information Processing Society of Japan