RK-005

協調学習によるソースコード識別子の命名学習方法
Collaborative Learning Method for Identifier Naming in Source Codes

式見 遼† 松浦 佐江子†
Ryo Shikimi Saeko Matsuura

1. はじめに

多くのソフトウェア開発プロジェクトにおいて、有用なライブラリやフレームワークを用いてプログラムが開発されることが多い。ソフトウェア開発では、他人が書いたソースコードを読んだり理解したりすることは不可避であるため、ソースコード中の識別子に適切な名前を付けることが、ソフトウェアの理解性や保守性を高めるための重要な要素のひとつである。識別子に適切な名前を付けることでソースコートの可読性が向上することは広く知られており、多くの研究者が識別子の名前の良し悪しがソースコードの品質に与える影響を議論している[1,2,3,4]。

また、我々は PBL(Project Based Learning)による実践的なソフトウェア開発実習を 10 年間実施[5]しているが、開発プロジェクトメンバーの間で十分な共通認識を持つことはソフトウェア開発経験が未熟な学生にとっては難しい問題である。こうした共通認識の基本はソフトウェアを構成する識別子に依存するため、プログラミング初学者の段階から識別子に適切な命名をする訓練を行うべきである。しかしながら、識別子の命名の適切さを評価する基準はコーディング規約以外には存在しない。

本論文では協調学習環境を用いて識別子の適切な命名とは何かを教育する手法を提案する。ここでは、コーディング規約では評価できないソフトウェアの概念を表す識別子の命名の適切さを、命名結果の多数決による評価を学習者に与え、比較させる。具体的には学習者に、ソースコード中の難読化された識別子の名前を問う問題を出題する。難読化とは識別子の名前を特に意味のないあるいは画一的な別名に置換することであり、これは学習上の狙いに従っていくつかの識別子を対象に行う。学習者は難読化されたソースコードを実行し、動作を理解した上で難読化された識別子の適切な名前を考えることによって命名技能を訓練する。協調学習環境は、全学習者の解答を収集し、それを学習者に公開する。学習者は解答結果を自分の解答と比較検討することで、自分の命名の適切さを評価することが出来る。

以下に本論文の構成を記す。本論文は 2 章で識別子の命名学習における問題点を、関連研究との比較により議論する。3 章では、本手法のインストラクショナルデザインを示す。これに基づく命名学習環境を 4 章で述べる。5 章では本手法の事例を示し、実験により学習者から得られた解答を評価し、本手法の有効性を議論する。

2. 命名学習における問題点

Butler ら[3]は不適切な名前の識別子がソースコードの品質に与える影響を不具合発見ツールの 1 つである FindBugs のデータを基に統計的に示している。彼らは適切な名前の識別子はプログラミング言語の命名規約に決して違反しないという見解を示しており、その上で、識別子名は開発するアプリケーション領域における概念および役割を適切に表すべきだとしている。コーディング規約はプログラミング言語ごとにソースコードの可読性を向上するために定めたガイドラインであるが、例え、ソースコードがコーディング規約を満たしているとしても、識別子の名前が開発アプリケーションの概念を適切に表していなければ、ソースコードの可読性は低下すると考えられ、コーディング時に、コード概念に基づいて識別子の名前の適切さを判断する基準を決定することが出来るかどうかが大きな課題である。

ソフトウェア開発では、顧客、アナリスト、設計者、プログラマーといったソフトウェア開発に携わる全ての人が、開発するソフトウェアに対する共通理解を持つ必要があり、最終的にはソースコードの識別子の名前で、この共通認識は表されることになる。Rajlich ら[4]はソースコードに現れる概念がプログラムを理解する上で重要であることから、ソースコードからアプリケーションドメインを学習する過程を提案しており、ソースコードは我々がアプリケーションドメインやソフトウェア設計が理解できるように書かれていることが望ましいことを示している。これは、言い換えると、プログラマーがソースコードのアプリケーションドメインを理解できるようにするには、コーディングを学習する前段階で識別子の命名を学習し、自分自身で適切な命名を判定できるような識別子命名学習環境を開発することが必要だということである。

Relf[1]はコーディング時に識別子の適切さを評価し、評価結果をプログラマーにフィードバックするツールを提案した。このツールによる評価はコーディング規約のみによるもので、概念の観点による適切さは判定できないという問題がある。

Deisenböck ら[2]は概念と名前の全単射写像により定義された形式モデルに基づいて簡明かつ一貫した命名を行う「ルール」を提案した。しかしアプリケーション領域のすべての識別子に適応できるわけではないし、ルールを構築する労力は大きい。

D. Lawrie[6]は省略形の識別子の名前が開発領域への理解に与える影響を議論した。それによると、非省略形の名前が最も良い理解を得ることが分かったが、その一方で、ほとんどの場合において省略形と非省略形の間に理解度の統計的な違いは見られなかった。

これらのことから、ソースコードの可読性を向上させるために最も重要なことは、ソフトウェアの開発に携わる多くの人が共通な理解を持つために、共通認識となる識別子の命名とは何かを訓練することである。しかし、共通認識とは、ソフトウェアに登場する一般的な数学等の常識、コンピュータサイエンスの基礎知識、アプリケーションドメインの知識、設計のノウハウ等多くの知識であり、一般化やすべてを定義してその妥当性を測ることは困難である。

† 芝浦工業大学 大学院理工学研究科 電気電子情報工学専攻

協調学習[7]は参加者の間で共通な理解を得ることに効果があることが知られており、我々は学習者に多数決による命名の評価を 1 つの共通認識の指標として提供する協調学習環境を用いて、プログラマーに適切な識別子の命名を訓練する方法を提案する。この学習手法では、教授者は難読化コード生成ツールを用いて、学習目標に合わせてソースコード中の数種類の識別子を難読化する。学習者は学習環境 WebStudy[8]上で、難読化ソースコードを実行しソースコードの概念を理解した上で、適切な識別子の命名を行い、識別子の命名を訓練する。学習者が自分の命名を解答した際に、WebStudy は、自動的に全学習者の解答結果を収集し、これを学習者に提供する。これにより、学習者は自分の解答と他の人の解答を比較し、よりよい命名を検討する。また、教授者の手によって学習者の解答から、アプリケーション領域の概念を適切に表した命名を選び、その結果を学習者に提供することで共通認識を深める。

3. 命名学習のインストラクショナルデザイン

本章では、識別子の命名練習のインストラクショナルデザインの過程を示す。

3.1 出題問題の設計

ソフトウェア開発において、出来る限り早く、そして正確に要求の変化に対応するために、成果物が要求の追跡可能性を持つことが必要である。すなわち、識別子の命名の質は、ソフトウェア開発のすべての過程の成果物の可読性に影響を及ぼす。

オブジェクト指向パラダイムは、ソフトウェア開発の全行程において「オブジェクト」が成果物の基本概念を定義するため、成果物の追跡可能性を維持することができる。そのため、識別子に適切な名前をつける訓練を行うことで、初心者オブジェクト指向プログラマーはよりオブジェクトの役割や概念を理解できるようになり、モデリングとデザインの技術を向上させることにつながる。

一般に識別子の名前が適切であるかを判定することは難しい。そこで、我々はプログラミング作法に関して有名な書籍[9,10]を参考に、オブジェクト指向パラダイムにおける適切な識別子名の規約を以下の様に定義した。

- クラス構造における識別子の命名の適切性
  クラスは複数のフィールドとメソッドから構成されるため、妥当性は以下の点から判断される。
  - フィールドの集合はデータの観点から、クラスの責務を表す
  - アクセス修飾子が "public" や "protected" なメソッドはクラスの責務を表す
  - アクセス修飾子が "private" なメソッドはクラス内でのみ使用される手続きの部分的な役割を表す
  - メソッドの手続きはその機能を適切に表す
  - メソッド内の一時変数とパラメータはメソッド内部での役割を表す
- 関連するクラス間における識別子の命名の適切性
  - 継承関係を持つクラスの間で、クラス名は親子関係を表す
  - 抽象クラスと具象クラスやインタフェースと実装の関係を持つクラスにおいて、両方のクラスは異なる抽象度を表す

3.2 プログラミング学習環境「WebStudy」

WebStudy はブラウザ上で動作する web ベースプログラミング学習環境である。GWT（Google Web Toolkit）で実装した C 言語と Java に対応した IDE（統合開発環境）を備え、ブラウザ上でコードの記述やコンパイル、実行を行うことができる。このため、WebStudy でのプログラミング学習ではインストールや初期設定といったプログラミング初学者の壁となる面倒な作業は必要ない。また、教授者に対してはプログラミング学習のための問題作成から、学習状況管理までの支援機能を提供している。教授者は WebStudy 上で説明と問題の組からなるコースを作成して学習者に提示し、その回答を収集して分析することができる。教授者はソフトウェア開発工程で遭遇する様々なシナリオに沿ってストーリーを定義し、学習者はプログラムを動作させながら、そのストーリーを学ぶ。ストーリーには、穴埋め式問題、選択式問題、コード問題等を設定できる。コード問題では、実際にコードを実行しながら、問題を解くことができるが、単なる実行だけではなく、拡張アドオンを動作させることで、コードの品質の評価や、特定のコード生成を行うことができる。

本提案では、「難読化コード生成ツール」が WebStudy のコード問題として動作する命名学習の問題と評価のエンジンを生成する。この評価エンジンは学習者の解答履歴を保持し、同じ問題を解いている他の学習者の解答を収集する。そして、収集結果を一覧として、学習者に提示する。学習者は自分の解答と他の学習者の解答を比較し、自分の命名が他の人の認識と共通しているか否かを判断し、その解答を修正することもできる。

3.3 命名学習プロセス

WebStudy による協調学習のプロセスは以下のとおりである。

1) 教授者は図 1 に示す難読化コード生成ツールを用いて、元のソースコードに対し、学習項目ごとに難読化すべき識別子を指定して、難読化を行い、問題を生成する。
2) 教授者は WebStudy のストーリーの中で、コード問題に、難読化コード生成ツールが生成した問題を登録する。また、WebStudy 上でこの問題に対して、「識別子置換アドオン」と「解答収集アドオン」を拡張機能として設定する。
3) 学習者は WebStudy のこの問題をプログラムを動作させながら、難読化された識別子に対して適切な名前を解答する。
4) 「解答収集アドオン」が同じ問題に対する回答を全て収集し、一覧として表示する。一覧には解答者の初めの命名と直近の解答の命名が表示され、それぞれの数が合計されている。他の学習者の命名と比較することで、同じソースコードの理解の仕方を学習し、より適切な命名を再度解答することができる。

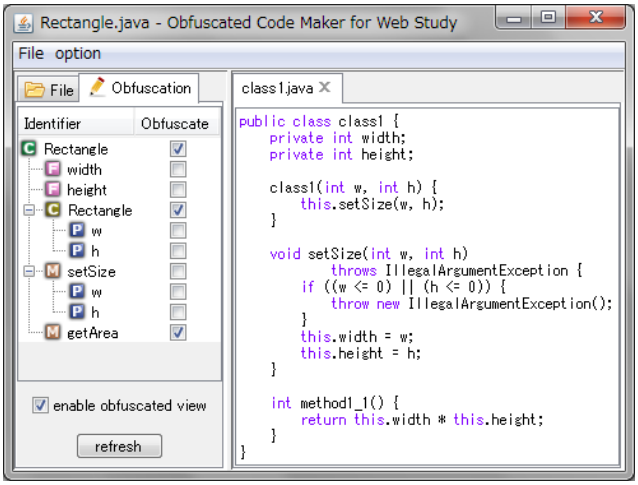

図 1 難読化コード生成ツール

4. 命名学習環境

本手法における識別子の命名学習は、学習問題の出題、学習者の解答、解答の評価からなる。本節では、これらの学習過程を順番に説明する。

4.1 問題の作成－難読化コード生成ツール

教授者は課題とするソースコードについて、任意の識別子の名前を置換したソースコードを作成する。この操作を難読化と呼ぶ。難読化したソースコードはコンパイル・実行が可能で、その挙動はオリジナルと全く同じである。

生成した難読化コードは「適切な識別子名を考える問題」として、学習者に出題する。

ソースコードの難読化は専用の難読化コード生成ツールを用いてクライアント環境で行う。難読化したソースコードは学習問題となるため、難読化を行う際には以下の点に気を付けなくてはならない。

- 対応する識別子の名前を一括で置換し、置換後のソースコードに、動作の変化やコンパイルエラーが起きないようにする
- 過度に難読化せずに、学習者がソースコードの構造や実行結果を見て、難読化された識別子の役割を理解できるようにする

専用の難読化コード生成ツールはこれらの問題を解決しながら難読化コードが生成出来るように設計されており、具体的には次の機能を備えている。

- 同一識別子の一括難読化
- 難読化ソースコードのプレビュー

ソースコードの難読化が完了したら、難読化コード生成ツールが生成するファイルを WebStudy にアップロードす

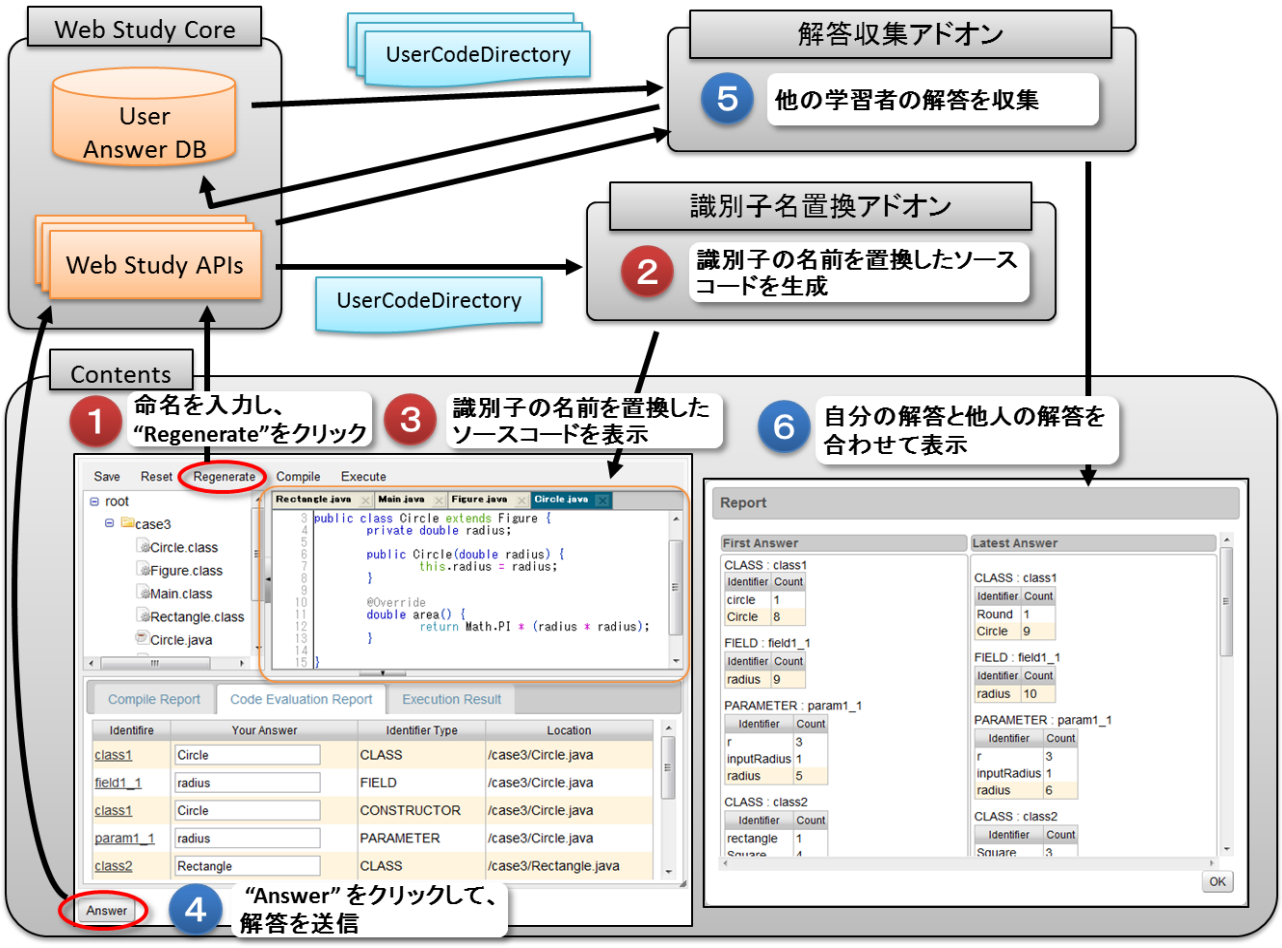

図 2　学習フロー

ることで、難読化された識別子の適切な名前を問う問題が WebStudy 上に展開される。

4.2 学習者による解答

学習者は WebStudy 上で、出題された難読化コードに対し、コードの構造や実行結果から、各識別子について適切だと思われる名前を推測して解答する。

学習者による解答、すなわち識別子の名前の入力は WebStudy の IDE 上で行う（図 2.①）。WebStudy の IDE は標準機能であるコードのコンパイル、コードの実行、と我々が開発した「識別子名置換アドオン」（図 2.②）が可能にする識別子の名前の置換の機能を提供する。これにより学習者はソースコードの動作理解と識別子の命名を考えることに集中して、問題に取り組むことが出来る。命名を終えた学習者が WebStudy 上で識別子の名前を確定し、解答を送信する（図 2.④）と、その解答は WebStudy によって学習者ごとに収集される。

4.3 解答の評価－命名の収集

識別子の適切な命名とは、他の人がソースコードを見たとき、その識別子の役割を理解できる名前である。逆に言えば、多くの人が定義した名前は、少数の人が定義する名前よりも命名として適切であることが多いと考えられる。

そこで、本手法では全学習者のソースコードから定義された識別子名の集計を行う。集計結果からわかる、多くの学習者が定義した識別子名は、多くの人がその識別子の役割を理解できるという意味で、適切だと考えられる。また、この収集結果を学習者に公表することで、学習者が一般的に適切である命名を直感的に理解する効果が期待できる。教授者は収集結果によって、識別子の一覧の何件かを適切な命名であると評価したり、適切でない理由を示すといった指導を行うことができる。

解答の収集は「解答収集アドオン」が行う（図 2.⑤）。「解答収集アドオン」は WebStudy システム内部のユーザ回答データベースを走査して、全学生の識別子の命名情報を収集する。

学習者の命名技量に対して難読化が難しすぎた場合や学習者の人数が少ない場合、良い命名が多数派にならないことがある。この場合、教授者は収集結果を次に出題する難読化コードの難易度に反映させて再度学習を行うことで、学習者に合った命名学習を構築できる。

収集された命名は識別子ごとに、ある名前をつけた学習者が何人いるのかを集計して表示・公開される（図 2.⑥）。この結果から各学習者は多くの学習者が定義した識別子名を良い手本として、少数の学習者が定義した識別子名を悪い手本として見ることで、識別子に適切な名前をつける能力を養うことができる。

このように、web 上で命名学習を行うことで、教授者の負担を極力減らしつつ学習を行うことができる。

5. 学習実験

3 章で既に述べたように、難読化ソースコードによる命名の学習は、題材となるソースコードによっては、データ構造の概念やプログラミングノウハウの理解などの命名学習以外の側面を持つ。本章では学習観点の異なるいくつかのソースコードを示し、そのソースコードを使って行った学習実験の結果を示して議論する。

5.1 学習問題例

実験を行うにあたり、我々は表 1 に示す学習上の狙いが異なる 6 つの学習問題を用意した。このうちの 2 つ、問題 3 の実験結果を 5.1.1 節で、問題 4 の実験結果を 5.1.2 節でそれぞれ示し、本手法の学習効果を議論する。

表 1 学習項目とねらい

	学習項目	ねらい
1	スタック：データ構造の概念	振る舞いの観点からデータ構造のスタックを見抜き、クラスとメソッドに適切な名前を付けることが出来る。
2	複素数： 数学の概念	数学の概念である複素数クラスのフィールドに適切な名前を付けることが出来る。
3	矩形：数学の概念（5.1.1 の例題）	クラス構造の観点から数学の概念である矩形を見抜き、クラスとフィールドに適切な名前を付けることが出来る。
4	似た役割を持つ変数（5.1.2 の例題）	対の役割を持つ局所変数に対して、それらを区別する適切な名前を付けることが出来る。
5	ビルダーパターン：デザインパターン	構造の観点からデザインパターンのビルダーパターンに気づき、クラスやメソッドに適切な名前を付けることが出来る。
6	コンポジットパターン：デザインパターン	振る舞いと構造の観点からデザインパターンのコンポジットパターンに気づき、フィールドやメソッドに適切な名前を付けることが出来る。

5.1.1 例 1.「オブジェクトの概念の理解」

オブジェクト指向言語は世の中に存在する様々な概念をクラスで構造化することができるが、プログラミング初学者にとって、概念を適切なクラス構造に分けることは容易ではない。その理由は、アプリケーション内で主たる役割を果たすオブジェクトの概念を捉えきれていないためである。このようなプログラミング初学者に対して、我々の提案する協調学習環境で難読化されたソースコードの識別子の命名を行うことで、オブジェクト指向の概念の理解が期待できる。

この具体例となるソースコードを図 3 に示す。このソースコードではクラスとそのフィールド、合わせて 3 つの識別子が難読化されている。そのため、学習者はクラス内の難読化されていない構成要素から、このクラスの概念を推測する。このクラスの唯一のメソッド *area()*はその名前からも、このクラスのインスタンスの面積を返すことは明らかである。そして、面積を求める式は *field1_1* * *field1_2* となっている。以上のことから、*class1*、*field1_1*、 *field1_2* に適切な名前は、Rectangle、width、height だとわかる。

また、*class1* は図 4 のソースコードから Figure クラスを継承する。この情報は問題のクラスが「面積を計算できる図形の一種である」ことを示しており、継承の概念に疎い初学者は、このソースコードから継承の具体例を学ぶことが出来る。

```
public class class1 extends Figure {
  private double field1_1;
  private double field1_2;
  ...
  @Override
  double area() {
    return field1_1 * field1_2;
  }
}
```

図 3 class1.java

```
public abstract class Figure {
  abstract double area();
}
```

図 4 Figure.java

5.1.2 例 2.「ローカル変数の命名」

役割の異なる別々の変数に似た名前を付けると、ソースコードの可読性は大きく損なわれる。プログラマーはメソッドやコンストラクタの局所変数に名前を付けるときは、このような問題が起きないように注意する必要がある。もちろん初学者にも、この種の問題意識は持たせるべきである。

図 5 は、可変長の int 型引数の総和を求めるメソッド sum のソースコードである。与えられる引数が必ず 1 つ以上になるように、固定長の引数 *param1_1* と可変長の引数 *param1_2* が定義されている。どちらも sum を計算するための引数であることに間違いないが、「最初の要素とその残り」という立ち位置が異なる両者には *arg0, arg1* などではない、両者の対の役割を明確に区別できる名前を付けるべきである。

```
public class Math {
  public static int sum(int param1_1,
      int... param1_2) {
    int sum = param1_1;
    for (int aRemainingArg : param1_2) {
      sum += aRemainingArg;
    }
    return sum;
  }
  ...
}
```

図 5 Math.java

5.2 実験結果と議論

我々は、表 1 に示す 5.1.1 節、5.1.2 節を含む 6 つの問題を使用して学習実験を行った。実験は java 使用歴 2 年程度の学生 9 人を対象とした。5.1.1 節の問題の実験結果を表 1 に、5.1.2 節の問題の実験結果を表 2 にそれぞれ示す。

表 2　実験 1 の結果

	初回解答		最終回答	
class1	識別子	人数	識別子	人数
	Square	4	Rectangle	5
	Rectangle	3	Square	3
	rectangle	1	Quadrangle	1
	Quadrangle	1		
field1_1	識別子	人数	識別子	人数
	height	4	height	5
	width	3	width	3
	length	2	length	1
field1_2	識別子	人数	識別子	人数
	width	6	width	6
	height	3	height	3

表 3　実験 2 の結果

	初回解答		最終回答	
param1_1	識別子	人数	識別子	人数
	firstNum	1	firstNum	1
	s	1	firstValue	1
	start	1	init	2
	firstValue	1	headNumber	1
	init	1	firstTerm	1
	headNumber	1	augend	1
	firstTerm	1	initVal	1
	initVal	1	augent	1
	augent	1		
param1_2	識別子	人数	識別子	人数
	nextTerm	1	nextTerm	1
	additionNum	1	additionNum	1
	restNumbers	1	restNumbers	1
	addValues	1	addValues	1
	addends	1	addends	2
	n	1	add	2
	add	1	backwardNums	1
	backwardNums	1		
	sumNum	1		

最初の実験では、学習者は概ねソースコードの概念を理解して命名ができている結果となった。初回解答時と最近の解答時では *class1* に Square と命名した学習者が減り、Rectangle と命名した学習者が増えており、他人の命名を参考にしてより適切な名前への修正が行われたことがわかる。

2 つ目の実験では、学習者ごとの命名が大きくばらついた。我々の想定する *param1_2* の模範解答は remainingArgs であり、これに類する解答をした学習者は restNumbers と backwardNums の 2 人のみであった。このように解答に大きなばらつきが出た場合は、出題の意図が理解されていないことがあるので、教授者の指導によるフォローが必要になると考える。また、我々の想定外の命名に addends があるが、この解答も妥当だと考えられる。我々の想定しない妥当な解答に票が集まったことに、この評価方法が「共通の理解を得る」という適切な識別子の命名の判定に正常に機能していることがわかる。

実験全体の傾向では、問題 1 から問題 3 では多くの学習者が識別子に同じ名前を付け、問題 4 から問題 6 ではほとんどの学習者がそれぞれ異なる命名を行った。この問題 4 から問題 6 のように、学習者の解答がばらついて収束しなかった場合、教授者は解答がばらついた理由を分析し、そ

の理由にあった指導を学習者に行う。今回の実験で学習が上手く行かなかった原因の一つは、学習者がソースコードのコアの概念を知らなかったことである。例えば問題 3 の「長方形」の概念はよく知られていたが、問題 5 の設計ノウハウの一つである「ビルダーパターン」を知っている学習者はほとんどいなかった。このような場合、教授者の行う指導としてはソースコードのコアの概念を教えることが有効である。また、問題 4 は多くの学習者がソースコードを理解できているにも関わらず命名がうまくできていない。このような場合は、学習者の解答の内いくつかの解答を評価して、その評価理由を学習者に示すことができる。いずれも、指導のあと再び同じ問題を解かせて改めて命名の学習ができる。また、問題を出題する前に予め概念を十分に理解させる講義を行うことも有効である。

本手法では、このような指導によって識別子の命名の勘所が得られるだけでなく、学習者の知らない新しい概念を、具体例をもって学習することができる。また、本実験では発生しなかった現象ではあるが、適切ではない命名が学習者の間での多数を占める場合が考えられる。このような場合も、教授者による指導や評価によって適切な解答へ学習者を導くことができる。

6. まとめ

本論文では多数決による命名の評価を提供する協調学習環境を用いて、適切な識別子の命名を学習する手法を提案した。また、我々の手法の効果を実験により検証し議論した。今後は、多数決による命名の評価と並行して命名規則による命名の評価を行うことを検討するとともに、より多くの学習者の下で実験を行いさらなる効果の検証を行う。

謝辞

本研究は文部科学省平成 21 年度大学教育・学生支援推進事業【テーマＡ】「工学系技術者のソフトウェア開発技能育成」の一環として実施した。

参考文献

[1] P.A. Relf, Tool Assisted Identifier Naming for Improved Software Readability: An Empirical Study, Proc of Int'l Symp. Empirical Software Eng., pp.53-62, 2005.
[2] F. Deisenböck and M. Pizka, Concise and consistent naming, Proc of the 13th International Workshop on Program Comprehension (IWPC 2005), IEEE Computer Society, pp.1-10,2005.
[3] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, Relating identifier naming flaws and code quality: an empirical study, Proc of WCRE'09, pp.31-39, 2009.
[4] V. Rajlich and N. Wilde, The Role of Concepts in Program Comprehension, Proceedings of IWPC 2002, pp. 271-278, 2002.
[5] 松浦佐江子，実践的ソフトウェア開発実習によるソフトウェア工学教育，情報処理学会論文誌，Vol.48，No.8，pp.2578-2595, 2007.
[6] D. Lawrie, C. Morrell, H. Field, and D. Binkley, What's in a Name? A Study of Identifiers, Proc. of the 14th IEEE International Conference on Program Comprehension (ICPC'06), pp.1-10,2006.
[7] Dillenbourg, P. (1999). Collaborative Learning: Cognitive and Computational Approaches. Advances in Learning and Instruction Series. New York, NY: Emerald Group Publishing Limited, United Kingdom.
[8] Web Study, http://www.sayo.se.shibaura-it.ac.jp/incusphere/webStudy.html
[9] Brian W. Pike, Rob Kernighan. (1999). The Practice of Programming. Addison-Wesley Professional
[10] Bloch, Joshua. (2008). Effective Java Second Edition. Addison-Wesley.