ローマ字入力時の日本語識別子入力補完プラグインの開発

熊谷 優斗　伊藤 恵　奥野 拓

本研究では，日本語識別子入力時のプログラマにかかる負担を軽減するプラグインの開発を行う．近年のソフトウェア開発プロジェクトの複雑化は，ソフトウェアの保守にかかる時間的コストを増大させており，コスト軽減のためにソースコードの可読性を高めることが重要視されている．ソースコードの可読性を高める方法の一つとして，ソースコード中の変数や関数の識別子名を日本語で書くことが有用であるとした報告がいくつか存在する．しかし，識別子を日本語入力する際には，文字変換が面倒であることや入力補完が効き難いことが問題があり，プログラマに入力の負担が発生してしまう．本研究では，ローマ字入力であっても日本語識別子のインクリメンタルサーチによる入力補完を実現するプラグインの開発を，日本語インクリメンタルサーチツールである Migemo を用いた手法と形態素解析を用いた手法の 2 種類によって試み，問題の解決を図る．

In this research, we develop a plug-in that reduces the burden on programmers when input Japanese identifier. The complication of software development projects in recent years has increased the time cost of maintaining software. So it is important to increase the readability of source code in order to reduce costs. There are several reports that it is useful to write the identifier names of variables and functions in Japanese as one way to increase the readability of source codes. However, when input Japanese identifier, there are some problems to struggle for programmers: the character conversion is troublesome and the input completion is not effective. In this research, we attempt two kinds of methods, one using Migemo and one using morphological analysis, to develop a plug-in that realizes incremental search for Japanese identifiers even if Romaji input mode, and to solve above problems.

1　はじめに

近年ではソフトウェア開発プロジェクトの大規模化や複雑化は，ソフトウェアの保守にかかる時間的コストが増させている [1]．保守作業において最も時間のかかる作業は，既存のソースコードを読み解き，理解することであると言われている [3]．そのため，ソースコードの可読性を高めることは重要であるとされ，これまでに様々な研究が行われている．Buse らはソースコードの可読性を高めるとされている数々の手法について，どれほどの効果があるのかを定量的に評価した [2]．その知見の一例としては，ソースコードの識別子の長さはそれほど可読性に影響しないことが挙げられる．このことから，識別子を省略して記述するより，意図が正確に伝わるような長い識別子を用いるべきであると言えると述べている．このように，ソースコードの識別子を適切に名付けることは可読性を向上させる効果があることが分かっている．

また，一般的なプログラミング言語では識別子の命名に基本的に英語を用いているため，英語が母国語ではない日本人にとっては，表現し辛い概念や用語を英語で表現しなければならない場合がある．識別子に日本語を用いることによって適切に表現できるようになり，ソースコードの可読性が向上するとして，日本語識別子の有用性を評価した研究がいくつか存在する．その先駆けとなった中川らは，日本語識別子を用いることでプログラムの読解に要する時間を短縮することができたと述べている [7]．このように，日本語識別子を用いることによって，ソースコードの可読性を

Development of a Plug-in that Makes Input Completion of Japanese Identifier Even If Romaji Input Mode
Yuto Kumagai, 公立はこだて未来大学システム情報科学研究科, Graduate School of Systems Information Science, Future University Hakodate.

向上させる効果があることは判明している．

上記の他に，日本語識別子を用いることによる利点としては以下が挙げられる．

- 識別子名を英語で考える必要がないため，日本語から英語へと翻訳するためにかかる時間が減る．テストメソッドなど，識別子名に詳細な意図を持たせたい場合は特に時間短縮となる．
- 仕様書からの詳細化が円滑に行える．特に，仕様書で用いられている単語をそのまま識別子名として用いることができるため，保守作業時に仕様書との対応が理解しやすい．

識別子に関して，識別子を考える，書く，書いた識別子を他者が読むという観点がある．このうち，日本語識別子を用いることによって識別子を考えることと他者が読むことにはメリットがあると述べた．しかし，日本語識別子を書く際に考えられる以下の課題については解決されていない．

- 識別子名として日本語を入力する際に，かな入力を ON にする，読みを入力する，読みを漢字表記へと変換する，目的の漢字へと変換する，入力を確定するというステップを踏む必要がある．日本語識別子を入力する度にこれらのステップを踏むことはプログラマに煩わしさを感じさせる可能性がある．
- 一般的な統合開発環境では，IME が ON になっている状態ではインクリメンタルサーチによる入力補完が効かない．よって，日本語識別子の呼び出しを行う際に，再度入力を行う必要があることや，識別子の型や引数の数が分からないといったことが問題点として挙げられる．

ここで，インクリメンタルサーチとは，1 文字入力するたびに検索を進め，その結果を動的に出力する検索手法のことである．通常は，キーワード全体を入力し，その内容を検索するが，インクリメンタルサーチは，ユーザが最初の 1 文字を入力した瞬間から検索を行う．本研究では，このインクリメンタルサーチによる入力補完を，日本語識別子に対しても行えるようにすることによって日本語識別子入力にかかる負担を軽減し，日本語識別子をより扱いやすいものとすることを目的とする．

本稿では，2 章で日本語識別子に関連する研究や，本手法で用いるツールについて説明する．3 章では日本語識別子を用いる際に被験者がどう感じるか調査するために行った実験の手順や結果について述べる．4 章では，本研究で開発するプラグインの 2 種類の手法について述べる．5 章では，本稿のまとめと今後の展望を述べる．

2　関連研究

2.1　日本語プログラミング言語と日本語識別子

これまでに,「和漢」や「なでしこ」といった数多くの日本語プログラミング言語が研究，開発されている [4] [5]．日本語プログラミング言語は，プログラムの文法を自然な日本語として読めるように記述することが特徴である．これにより日本人にとって読解が容易であり，プログラミング初学者であってもプログラムの内容を大まかに理解することができるというメリットがある．一方で，他の一般的なプログラミング言語とは形式が違う独自の書式を学ぶ必要があることが課題として挙げられる．

これに対し，日本語識別子は文法の記述はそのままに，識別子の記述のみを日本語で行う．これにより，プログラミング言語に対する新たな知識は必要とせず，適切な命名を行える場合がある．現在主流であるほとんどのプログラミング言語において，2 バイト文字を識別子として用いることが可能である．日本語識別子についての研究として，平田らが日本語のみで書かれた識別子を用いることによる，可読性の変化を評価する研究を行った [8]．この研究では，100 行程度で構成される 4 種類のプログラムを日本語識別子及び英語識別子で書き，プログラム内にバグを潜ませておいた．それらを学部 4 年生と大学院生計 18 人にデバッグしてもらい，かかった時間を計測した．結果として，4 種類中 3 種類のプログラムにおいて，英語識別子より日本語識別子の方が早くデバッグが終了する結果になった．このことから，日本語識別子を用いることで，英語識別子よりも可読性が高まったとした．また，中川らは日本語識別子を用いて相当規模のソフトウェア開発を行った [7]．その中で，日本語識別子によって保守性が高まったことを明らかにしている．

2.2 Migemo

日本語のインクリメンタルサーチを行うツールとして，Migemo が挙げられる [6]．Migemo では，ユーザのローマ字の入力 1 文字 1 文字に対し，指定された読みで始まる単語を正規表現に動的に展開することでインクリメンタルサーチを実現している．Migemo は Emacs や Vim などのテキストエディタの，日本語文字検索を快適に行うための拡張機能を実現するために用いられている．本研究で開発するプラグインの手法の 1 つとして，統合開発環境の内部で Migemo を動作させ，宣言済みの日本語識別子とのマッチングをすることでインクリメンタルサーチによる入力補完を実現する．

3 日本語識別子入力に関する調査

日本語識別子を入力する際に被験者がどのように感じるかを調査する実験を行った．図 1 に実験の様子を示す．対象は，日本語識別子を用いたことがない本学学生 5 名とした．これらの被験者は，本学の講義を通して一定以上のプログラミング経験を得た学生達である．本調査の仮説は，日本語入力のために複数のステップを踏む必要があることや，インクリメンタルサーチによる入力補完が効かないことが識別子入力の妨げとなることである．

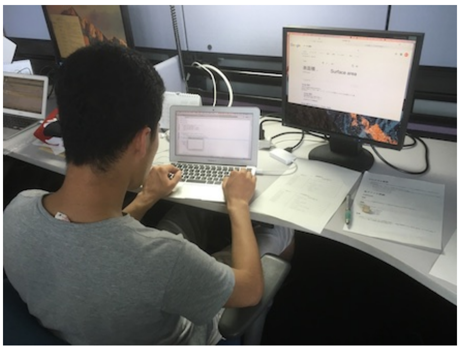

図 1　実験の様子

3.1 実験内容

予めこちらで簡単な計算アルゴリズムを処理するプログラムを作成した．プログラム内の識別子名は，被験者に命名してもらうために “method1” などの仮のものを命名しておいた．また，プログラムの読解を助けるために，日本語で書かれたプログラム仕様書を作成した．作成したプログラムと仕様書は付録 A に載せる．被験者にはまずプログラムと仕様書を読んでもらい，プログラムの内容を理解してもらった．その後，仮に命名した識別子に対し，第三者が読んだ際に理解できるような命名でという条件で，英語識別子名と日本語識別子名をそれぞれ命名してもらった．更に，仮に書かれた識別子を命名した識別子に入力し直す作業をしてもらった．この際，普段のコーディングと同様な条件で入力してもらうために，コピーアンドペーストや一括置換といった動作は制限した．これらのタスクが終了した後に，アンケートに答えてもらった．

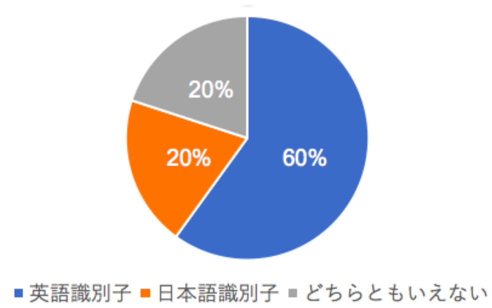

図 2　識別子入力し難かったのはどちらの言語か？

3.2 結果と考察

「識別子入力し難かったのはどちらの言語か？」というアンケートの結果を図 2 に示す．結果として，5 名中 3 名の被験者が日本語識別子であると答えた．回答の理由としては,「日本語識別子を入力しても入力補完が効かなかったため」「半角英数と日本語を混ぜた識別子にしたため入力が大変であった」などが挙げられた．これらの被験者は仮説同様の負担を感じていたと考えられる．一方で 5 名中 2 名の被験者は英語識別子もしくはどちらでもないと回答した．回答の理由としては,「日本語識別子を短い名前で命名したため入力に苦労を感じなかった」「関数を入力す

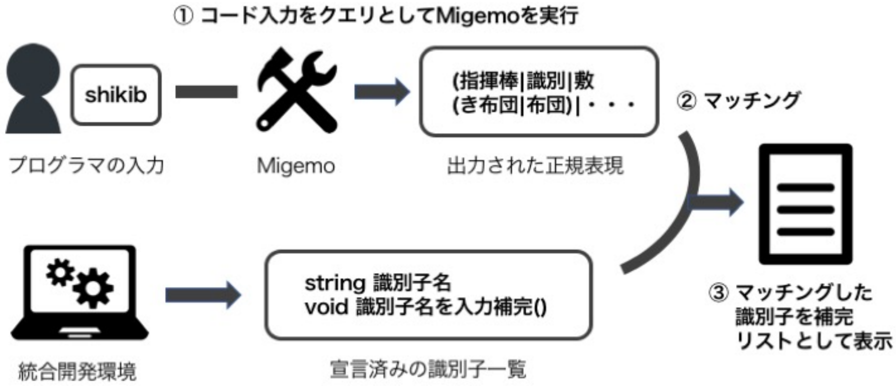

図 3　Migemo を用いた手法

る際に引数を気にする必要がなかったので苦労を感じなかった」といった内容であった．1 つ目の理由を回答した被験者に対して，入力数の多い日本語識別子を入力した場合どのように感じるか調査するため，別途単語数の多い日本語識別子を用意し，それを入力するタスクを行なった，その結果，入力に苦労したという回答が得られた．また，2 つ目の理由については，今回の調査では，関数の引数は予め入力してある状態であったため，このような回答が得られたと考えられる．一般的な統合開発環境の入力補完では，関数名を補完した際に引数の数，型も同時に分かるように補完がされる．入力補完をせずに日本語識別子を入力する際には，このような補完の恩恵を得ることもできないため，プログラマの負担は大きいと考えられる．

今回の調査を通して，多くの被験者が日本語識別子の入力の際に負担を感じたと分かった．また，入力補完を行わない場合の新たな問題点も明らかになった．

4　プラグインの作成

今回のプログラムは，GitHub 社が開発している統合開発環境である，Atom のプラグインとして開発を行う．Atom はオープンソースソフトウェアであり，機能の拡張が容易である．また，プラグインとして実装するため，Atom を用いているプログラマであれば誰でも実装した機能を導入することができる．

日本語識別子をローマ字入力によって入力補完するための手法として，次の 2 種類を提案する．

1. Migemo を用いた手法
2. 形態素解析を用いた手法

それぞれの手法について，次節以降にて詳細を述べる．

4.1　Migemo を用いた手法

日本語のインクリメンタルサーチを行うツールである Migemo を用いる．Migemo の詳細については 2.2 節を参照されたい．

この手法の概略は以下の通りである．また，概略を図示したものを図 3 に示す．

1. プログラマのコード入力をクエリとして Migemo を実行する．
2. Migemo から出力された正規表現と，統合開発環境内部に保持されている宣言済み識別子とをマッチングさせる．
3. マッチした識別子をコード補完リストとして表示する．

4.2　形態素解析を用いた手法

日本語識別子に対し形態素解析を行うことによって実現する．形態素解析とは，自然言語の文章を形態素ごとに分割し，それぞれの品詞や読み，活用形などを判別することである．

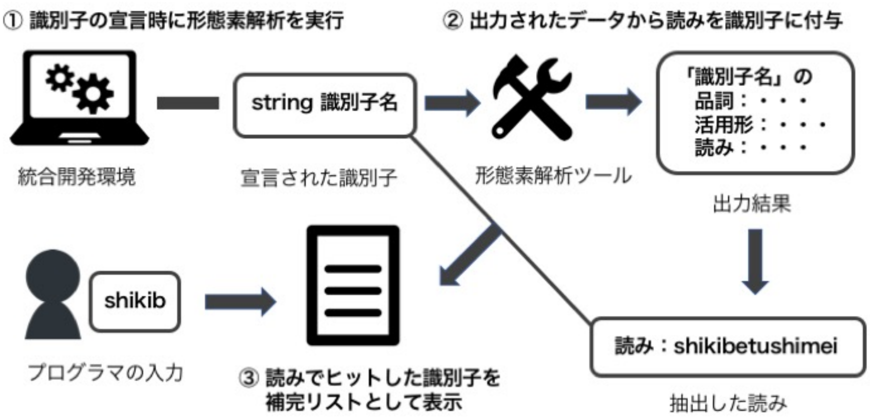

図 4　形態素解析を用いた手法

この手法の概略は以下の通りである．概略を図示したものを図 4 に示す．

1. 日本語識別子の宣言時に、識別子を入力として形態素解析ツールを実行する．
2. 出力されたデータのうち，ローマ字による読みを抽出し、元の単語とセットでメモリ上に保持する．
3. 入力補完時にはローマ字の読みで検索にヒットさせ、コードとして入力する際には日本語識別子を代入するようにする．

5　おわりに

本稿では，日本語識別子を実際に用いる際の問題点を論じ，調査した．結果として，日本語識別子を用いたプログラミングをする際に，日本語入力の手間が負担となっていることを明らかにした．また，それを解決するための 2 種類の手法を検討した．今後はどちらの手法がより適切であるかをプロトタイプを作成することで明らかにする．その後，作成したプラグインを用い，日本語識別子の入力にかかる負担が軽減されていることを実験によって明らかにする．

参 考 文 献

[1] Boehm, B. and Basili, V.R.: Software Defect Reduction Top 10 List, Computer, Vol.34, No.1, pp.135137 (2001).

[2] Buse, R.P.L. and Weimer, W.R.: Learning a Metric for Code Readability, IEEE Trans. Softw. Eng., Vol36, No.4, pp.546-558(2010).

[3] Goldberg, A.: Programmer as Reader, IEEE Softw.,Vol.4, No.5, pp.6270 (1987)

[4] 酒徳峰章: 日本語プログラミング言語「なでしこ」, コンピュータソフトウェア 28(4), 23-28, 2011-10-25.

[5] 鈴木孝則: 日本語プログラミング言語『和漢』, 情報処理学会研究報告計算機アーキテ クチャ(ARC), 1983(44(1983-ARC-029)), 1-10, 1983-11-28.

[6] 高林 哲, 小松 弘幸, 増井 俊之: Migemo: 日本語のインクリメンタル検索, 情報処理学会論文誌, Vol43, No.12, pp3968-3705 (2002).

[7] 中川 正樹, 早川 栄, 玉木 裕一, 曽谷 俊男: 日本語プログラミングの実践とその効果, 情報処理学会論文誌, Vol35, No.10, pp2170-2179 (1994).

[8] 平田篤志, 早川栄一, 並木美太郎, 高橋延匡: 識別子と内部コード系に着目した日本語によるプログラム の可読性の一評価, 情報処理学会研究報告ソフトウェア工学(SE),Vol. 1995, No. 55(1995), pp. 1-8.

A　調査に用いたソースコードと仕様書

```java
package math;

import math.Point;

public class myClass {

  public static double[] method1(Point p1, Point p2) {
    double[] variable1 = new double[3];
    variable1[0] = Math.abs(p1.getX() - p2.getX());
    variable1[1] = Math.abs(p1.getY() - p2.getY());
    variable1[2] = Math.abs(p1.getZ() - p2.getZ());
    return variable1;
  }

  public static double method2(Point p1, Point p2) {
    double variable2;
    variable2 = (method1(p1, p2)[0] * method1(p1, p2)[1]
          + method1(p1, p2)[1] * method1(p1, p2)[2]
          + method1(p1, p2)[0] * method1(p1, p2)[2]) * 2;
    return variable2;
  }

  public static double method3(Point[] points) {
    double variable3 = 0.0;
    double variable4 = 0.0;
    for(int i=0; i<points.length; i++){
      for(int j=0; j<points.length; j++){
        variable3 = method2(points[i], points[j]);
        if(variable3 > variable4){
          variable4 = variable3;
        }
      }
    }
    return variable4;
  }

  public static void main(String[] args) {
    Point[] pps = new Point[]{
        new Point(6.0,8.0,2.0), new Point(3.0,4.0,6.0), new Point(1.0,5.0,3.0), new
             Point(10.0,4.0,2.0)
        };
    System.out.println("与えられた座標から描ける直方体の最大表面積は" + method3(pps));
  }
}
```

リスト 1　**math.myClass**

```java
package math;

public class Point {

  private double[] points = new double[3];

  Point(double x, double y, double z) {
    points[0] = x;
    points[1] = y;
    points[2] = z;
  }

  public double getX() {
    return points[0];
  }

  public double getY() {
    return points[1];
  }

  public double getZ() {
    return points[2];
  }
}
```

リスト 2 **math.Point**

プログラム概要

- 3次元空間に与えられた複数の座標点から2点を選び出し、それらの座標を対角点とした直方体の表面積を求める。
- この処理を繰り返し、得られた表面積の最大値を出力する。
    - 例：(1.0,2.0,3.0),(4.0,3.0,1.0),(2.0,1.0,2.0)の3つの座標が与えられたとする。このときは以下の座標を対角点とした直方体が存在する
        - (1.0,2.0,3.0),(4.0,3.0,1.0)
        - (4.0,3.0,1.0),(2.0,1.0,2.0)
        - (1.0,2.0,3.0),(2.0,1.0,2.0)

各メソッド詳細

method1

概要

与えられた2つの座標点を対角点とした直方体の各軸上の辺の長さの値を返す。

変数

```
double[] variable1
```

- 各軸上の辺の長さを格納する
    - double[0] ... x軸の長さ
    - double[1] ... y軸の長さ
    - double[2] ... z軸の長さ

処理

- `point` クラスのゲッターから各座標を取得し、Javaの標準クラスである `math` クラスの `abs` メソッドにより軸上の距離の絶対値を取得する。

図 5　仕様書 1 ページ目

method2

概要

与えられた2つの座標点を対角点とした直方体の表面積の値を返す。

変数

```
double variable2
```

- 得られた表面積を格納する

処理

- 表面積の計算式は以下の通り。
    - 表面積 = 2(縦*横 + 縦*高さ + 横*高さ)
- 辺の長さは `method1` により取得する。

method3

概要

与えられた座標点から形成可能な全ての直方体の表面積を計算し、その中の最大値を返す。

変数

```
double variable3
```

- 現在の2つ座標点から形成された直方体の表面積を格納する

```
double variable4
```

- これまで得られた表面積の中の最大値を格納する

処理

- 2重for文により与えられた座標点から選び出せる2つの組を全て探索する
- 表面積は `method2` により取得する。
- 現在の座標の組から得られた表面積 `variable3` がこれまで得られた表面積の最大値 `variable4` より大きい場合、 `variable3` を `variable4` に代入する。

図 6　仕様書 2 ページ目