ローマ字入力による日本語識別子入力補完プラグインの開発と評価

熊谷 優斗　伊藤 恵　奥野 拓

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いる．保守作業において最も時間的コストがかかる作業は，既存のソースコードの読解作業であるとされ，コスト削減のためにソースコードの可読性を高めることが重要である．可読性向上の方法の 1 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はいくつか存在する．しかし，識別子を日本語入力する際には，文字変換の手間があることや入力補完が効き辛いことが原因で，プログラマに入力の負担が発生してしまう．本研究では，日本語識別子をより扱いやすくすることを目的として，ローマ字入力による日本語識別子の入力補完を可能とするツールの作成を行なった．また，作成したツールを用いて被験者に日本語識別子の入力を行なってもらい，入力の負担を軽減できたことを示した．

Recent years, the complication of software development projects has increased the time cost of software maintenance. It is said that reading work of existing source codes is the most time-consuming work in maintenance works, and it is important to increase readability of source codes for cost reduction. 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 for Japanese programmers. However, when entering the identifier in Japanese, the burden of input is occured to the programmer. There are two reasons for that. First, there is trouble of character conversion. Seconds, input complementation is hard to work.In this research, we created a tool that enables complementation of Japanese identifier by inputting romaji characters for the purpose of making Japanese identifiers easier to handle. In addition, we showed that we were able to reduce the burden of input by asking subjects to input Japanese identifier using the created tool.

1　はじめに

近年の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を増加させている [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 [4]．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．つまり，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と言える．

また，識別子に日本語を用いることによって日本人にとって英語で表現し難い概念や用語を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存在する．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 [9]．このように，日本語識別子を用いるこ

Development and Evaluation 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.

とによって，ソースコードの可読性を向上させる効果があることは判明している．

しかし，識別子名に日本語を用いることによって様々な問題が発生すると考えられる．例えば，初めて識別子が日本語で書かれたソースコードを読んだプログラマは読み辛いと感じる可能性が高いことが問題として挙げられる．また，変数名を日本語で宣言することによって，入力に関する問題が発生する．近年の一般的なエディタにはインクリメンタルサーチによる入力補完機能が搭載されている．この機能によってユーザは単語の一部を入力することで目的のコードを補完することができるため，入力時間の省略が行えるほか，型や引数の数なども同時に把握できるためコーディングミスを防ぐ役割があるとも考えられる．しかし，日本語変換がオンなっている状態では入力補完が効かないため，入力の手間やコーディングミスが発生し，プログラマに煩わしさを発生させてしまうと考えらえる．

本研究では，日本語識別子を用いることによって発生する入力に関する問題を解決し，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．アプローチとして，日本語識別子の読み仮名を形態素解析エンジンを用いることによって取得することによって，ローマ字入力のまま日本語識別子のコード補完を行えるようにするツールの開発を行う．

本稿では，2 章で関連研究について述べる．3 章では問題解決のアプローチについて述べる．4 章では実装した日本語識別子入力補完パッケージの全体の構成や処理の流れについて述べる．5 章では，開発パッケージを評価するために行った予備実験の内容と考察について述べる．6 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

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 [7] [8]．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問題として挙げられる．

2.2 日本語識別子

日本語プログラミング言語に対し，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，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 [10]．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を用い，プログラム内にバグを潜ませておいた．それらを学部 4 年生と大学院生合わせて 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した．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結論付けた．また，中川らは日本語識別子を用いて大規模のソフトウェア開発を行った [9]．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．

3 アプローチ

前述の通り，一般的なエディタでは日本語識別子をインクリメンタルサーチによって入力補完することができない．そこで，本研究では日本語識別子の読み仮名を保持することによってローマ字入力による日本語識別子の入力補完を可能にするプラグインの開発を行う．読み仮名は形態素解析エンジンを識別子名を入力として実行することによって，ヘボン式のローマ字の綴りを取得する．入力補完候補を提示する際には，エディタに入力されているローマ字と保持している読み仮名とをマッチングさせることによって，対応した

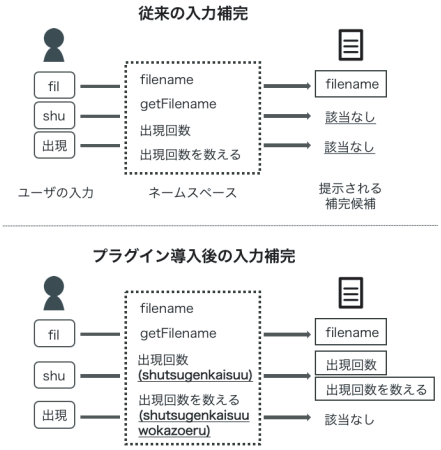

図 1　従来の入力補完と提案パッケージ導入後の入力補完

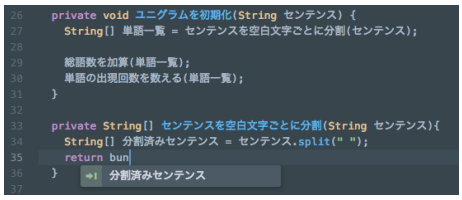

図 2　開発パッケージによる入力補完

日本語識別子の提示を行う．図 1 に従来の入力補完と提案パッケージ導入後の入力補完の様子を示す．

4　パッケージの開発

本章では，開発した日本語識別子入力補完パッケージの開発環境と処理の流れについて述べる．図 2 は開発パッケージを使用し，ローマ字入力によって日本語識別子の補完候補を提示している様子である．

4.1　対象とするエディタ

今回の日本語識別子入力補完プログラムは，Atom のパッケージとして開発を行なった．Atom は GitHub 社が開発したテキストエディタであり，オープンソースであるため第三者による拡張機能の開発が容易に行えることが特徴である [3]．Atom には入力補完機能を追加するパッケージがデフォルトでインストールされており，入力補完機能拡張のための API が公開されている．本研究ではこの API を利用した日本語識別子入力補完機能を開発した．

4.2　全体の構成と処理の流れ

全体の構成としては大きく分けて以下の通りである．図 3 に全体の構成と処理の流れを示す．

1. 日本語識別子抽出処理
2. 形態素解析エンジンによる読み仮名抽出処理
3. 入力補完候補提示処理

全ての処理は JavaScript を用いて開発した．処理の流れとしては次の通りである．

まず，パッケージの起動時に開いているソースコードから変数の識別子名を抽出する．それらの識別子名から，日本語が含まれているものを抽出し，それぞれの読み仮名の抽出を行う．読み仮名の抽出にはオープンソースの形態素解析エンジンである MeCab を用いる [6]．また，識別子名には固有名詞が使用される可能性を考え，辞書ファイルには固有名詞の解析に強い mecab-ipadic-NEologd を利用する [5]．これにより，固有名称の読み仮名の振り間違いを未然に防ぐ可能性が高くなると考えた．抽出した読み仮名はヘボン式ローマ字に変換し，元の識別子名と合わせて宣言済み日本語識別子リストとして保持する．例えば，“出現回数” という識別子名に対しては，“shutsugenkaisuu” という読みをキーとして保持する．

エディタ上に任意の文字が入力された際には，入力途中の文字列と日本語識別子リストとを前方一致で検索し，マッチしたものを補完候補として提示する．補完候補の提示には，Atom の入力補完パッケージ autocomplete-plus の拡張用 API である Provider API を利用する．Provider API のインクリメンタルサーチ機能によって，ユーザの入力に合わせて動的に補完候補を絞り込むことができる．

5　予備実験

開発パッケージによってプログラマの負担を軽減することが可能であるか調査した．図 4 に実験の様子を示す．対象は日本語識別子を普段用いることが無い本学の学生 8 名とした．学生は，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ており，実験

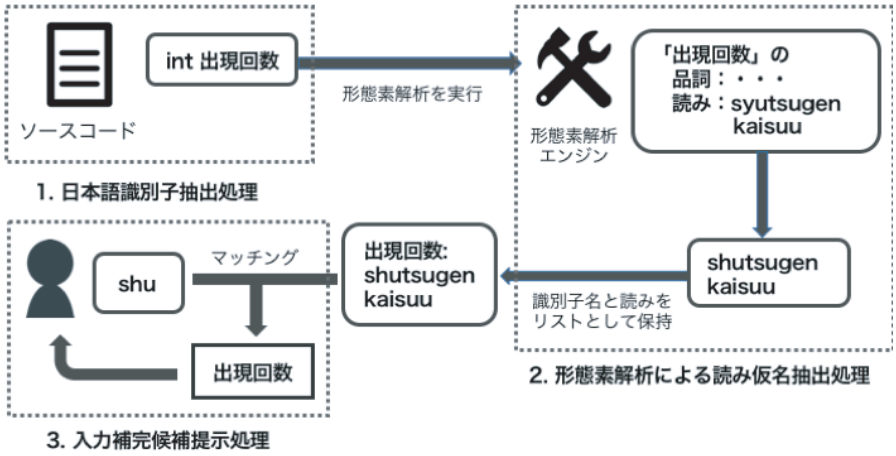

図 3　開発パッケージの全体の構成と処理の流れ

用プログラムで用いる Java についての知識も得ている．本実験では，プログラマの負担を軽減することが可能であるかを調査するほか，実験を行う上で開発したパッケージに不備はないかどうか明らかにすることを目的とする．

5.1　実験内容

予め実験用プログラムを作成した．作成した実験用プラグラムは付録 A に載せる．プログラムは簡単な計算を行うものを予め Java を用いて作成し，最大値を求めるアルゴリズムなど，一部の計算処理を除いたものを実験用プログラムとして扱った．処理を除いた箇所にはどのような処理を追加すれば正しく動作するかをソースコメントとして書いた．被験者には，実験用プログラムを読んでもらい，実験用プログラムを正しく動作させる上で不足している処理を追記してもらうタスクを課した．このタスクを，開発パッケージを用いない場合と用いた場合とで二度行ってもらい，タスク終了後はアンケートに答えてもらった．

5.2　実験結果

「パッケージ使用前は日本語識別子の入力の手間をどう感じたか」というアンケートの結果を図 5 に示す．質問に対して，「手間である」と感じるのであれば 1，「普段と変わりない」と感じるのであれば 4 という基準で，4 段階の値で回答してもらった．結果として，8 名中 5 名の被験者が 1 と解答し，2 名の被験者が 2 と解答した．つまり，大半の被験者が手間であると感じたことが判った．回答の理由としては，「日本語入力のオンオフによる全角記号の入力ミスが発生するため」「変換ミスをすると入力を取り消して入力し直さなければならないため」などが挙げられた．次に，「パッケージ使用前と比較して日本語識別子の入力の問題は改善したか」というアンケートの結果を図 6 に示す．質問に対して，「変わらない」と感じるのであれば 1，「ほとんど改善した」と感じるのであれば 4 という基準で，4 段階の値で回答してもらった．結果

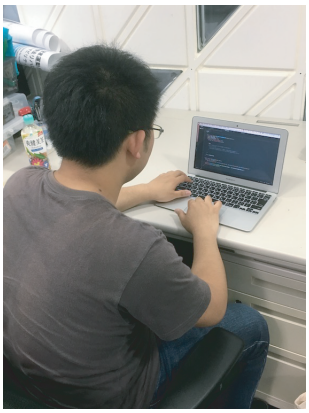

図 4　実験の様子

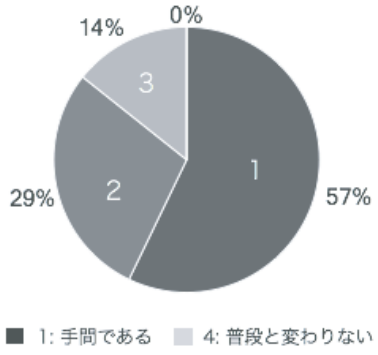

図 5　パッケージ使用前は
日本語識別子の入力の手間をどう感じたか

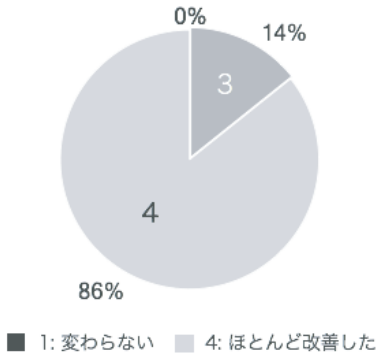

図 6　パッケージ使用前と比較して
日本語識別子の入力の問題は改善したか

として，8 名中 7 名の被験者が 4，8 名中 1 名の被験者が 3 と回答した．回答の理由としては，「かなに変換する手間を省けたのでスムーズに入力することができた」，「日本語入力のオンオフが不要になった点は大きいと感じた」などが挙げられた．これらの結果から，被験者がパッケージ使用前に感じた日本語識別子入力の問題はほぼ全て解決できたと考えられる．

5.3　振り仮名付与の課題

実験の最中にパッケージの動作に不備が発生した．被験者が「辺の長さ」という日本語識別子を用いた変数を宣言した際に，“hennnonagasa” という振り仮名が付与される想定であったが，“atarinonagasa” という振り仮名が付与されてしまった．これにより，被験者は目的の変数をうまく入力補完することが出来ないという問題が発生した．このような問題は他の入力においても発生する可能性があるため，誤った振り仮名付与に対しての修正機能を追加する必要があると考えられる．

6　まとめと今後の課題

本稿では，日本語識別子の入力に関する問題を解決するアプローチについて論じ，アプローチに基づいて行った Atom のパッケージの開発について論じた．また，開発したパッケージが日本語識別子を入力する際の問題を解決できているか評価するための実験を行った．結果として，大半の被験者が開発パッケージによって入力の問題が解決したと回答した．一方で，実験によって識別子に振り仮名を付与する際に誤った振り仮名を付与してしまう問題が発生した．この問題に対応するために，誤った振り仮名付与に対する修正機能の追加を行う必要がある．また，今回の実験では定量的な評価を行えていないため，パッケージ使用前と使用後とで被験者のコーディングミスの回数を計測するなどして，プログラマの負担が軽減できたことを定量的に評価する必要がある．

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] GitHub: Atom (Online), `https://atom.io` (Aug.5, 2018).

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

[5] Toshinori Sato: mecab-ipadic-neologd (Online), `https://github.com/neologd/mecab-ipadic-neologd` (Aug.5, 2018).

[6] 工藤拓: MeCab:Yet Another Part-of-Speech and Morphological Analyzer (Online), `http://taku910.github.io/mecab/` (Aug.5, 2018).

[7] 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.

[8] 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.

[9] 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).

[10] 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 実験用プログラム

```
import java.util.HashMap;
import java.util.Map;

public class Cuboid {

  public static HashMap<String, Double> 辺の長さを計算(Point p1, Point p2) {
    HashMap<String, Double> 長さ = new HashMap<String, Double>();

    double 縦の長さ = Math.abs(p1.X座標を取得() - p2.X座標を取得());
    長さ.put("縦の長さ", 縦の長さ);

    double 横の長さ = Math.abs(p1.Y座標を取得() - p2.Y座標を取得());
    長さ.put("横の長さ", 横の長さ);

    double 高さの長さ = Math.abs(p1.Z座標を取得() - p2.Z座標を取得());
    長さ.put("高さ", 高さ);

    return 長さ;
  }

  public static double 直方体の表面積を計算(Point p1, Point p2) {
    double 表面積;

    /*
    TODO:
    表面積を計算する
    表面積 = 2*(縦*横+横*高さ+高さ*縦)
    */

    return 表面積;
  }

  public static double 最大の直方体の表面積を計算(Point[] points) {
    double 現在の表面積 = 0.0;
    double 表面積の最大値 = 0.0;
    for(int i=0; i<points.length; i++){
      for(int j=0; j<points.length; j++){

        /*
        TODO:
        与えられた複数点から描ける直方体の表面積から最大値を計算する
        算出アルゴリズムの例:
          与えられた点から描ける直方体をループで探索し、"現在の表面積"に表面積を代
              入
          "現在の表面積"が"表面積の最大値"より大きければその値を"表面積の最大値"に
              代入する
        */

      }
    }
    return 表面積の最大値;
  }

  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
        " + 最大の直方体の表面積を計算(pps));
    System.out.println("#正しくは80.0");
  }
}
```

リスト 1　Cuboid.java

```
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 X座標を取得() {
    return points[0];
  }

  public double Y座標を取得() {
    return points[1];
  }

  public double Z座標を取得() {
    return points[2];
  }
}
```

リスト 2　Point.java