静岡大学情報学部情報社会学科 卒業研究

品詞に着目した識別子の命名方法の提案

鳥居　克哉（7071–1041）

2021年1月

指導教員：厨子 光政

卒業研究要旨

社会におけるソフトウェアの需要が高まるにつれ、ソフトウェアの品質がより重要になっている。ソフトウェアの品質にはソースコードの可読性が関係している。ソースコードの可読性を左右する要因は様々なものがあるが、その一つとして識別子名が挙げられる。一般に、識別子の命名には単語が用いられる。識別子名はソフトウェア開発者が自由に決めることができるが、その自由さが問題となり、ソースコードの可読性が低下することがある。それゆえ、しばしば命名規則が導入されることがある。この命名規則では識別子名の形式や、文字数等が定められる。また、品詞に関係する規則が定められることがある。これより、識別子の命名と品詞には関係があると考えられる。そこで本研究では、識別子の命名と品詞の関係を明らかにするために、javaによって記述されたオープンソース・ソフトウェアのプロジェクトを対象として、識別子の抽出と識別子名中の各単語の品詞の取得を行った。調査の結果より、識別子のほとんどが名詞及び名詞句になるように命名されており、メソッド識別子では、動詞または動詞句になるように命名される傾向があることがわかった。さらに、識別子の命名と自然言語において、副詞、形容詞、所有格の働きが共通していることがわかった。これらの関係をもとに、品詞に着目した命名方法の提案を行なった。

目次

序論

インターネットが我々の生活にとって欠かせない存在になるにつれて、ソフトウェアは社会の様々な場面で活用されるようになった。それに伴い、ソフトウェアに対する要求は増大し、ソフトウェアは大規模化や複雑化が進んでいる。そして、ソフトウェアの大規模化や複雑化に伴い、ソフトウェア保守にかかるコストが増大するという問題が発生している。

一般に、ソフトウェア開発は複数のソフトウェア開発者間でソースコードを共有する。それゆえ、ソースコードの可読性はソフトウェア保守のコストに影響を与える因子の一つである。可読性もまた、様々な因子によって成り立っている。その因子の一つに識別子が挙げられる。

ほとんどのプログラミング言語において、識別子はソフトウェア開発者が自由に命名することができるが、自然言語中の単語を用いて命名されることがほとんどである。また、ソフトウェア開発プロジェクトによっては、命名規則が導入されることがあり、しばしば品詞に関する命名規則が存在する。

そこで、本研究では、オープンソース・ソフトウェアを対象として調査により、識別子中の品詞の関係を明らかにし、品詞に着目した識別子の命名方法を提案する。

本論文の構成は次の通りである。まず第一章では、ソースコードの可読性に関係すると考えられる要素や、プログラミング言語の機能など、研究の背景について述べる。次に第二章では、オープンソース・ソフトウェアから識別子中の単語を抽出し、品詞を取得する方法について述べる。第三章では調査によって得られたデータの概要や、調査結果を基にした識別子名中の品詞の関係の考察について述べる。そして、考察を基に、品詞に着目した命名方法の提案とその方法の評価を行う。

第一章　研究の背景

本章では、研究の背景について述べる。

1.1　ソースコードにおける可読性

可読性とは、テキストがどれだけわかりやすいかという人間の判断のことである[1]。これより、ソースコードにおける可読性(Radability)とは、ソースコードによって表現されたプログラムの内容が、人間にとってどれだけわかりやすいかという指標であると言える。

ソースコードの可読性に影響を与える因子は数多く存在する。例えば、ソースコードのインデントや、ソースコード中のコメントなどが挙げられる。ソースコードにおけるコメントは、ソースコードへの注釈であり、プログラムの実行結果に影響を及ぼさない。また、Pythonのように、インデントによるコードブロック形成を行わないプログラミング言語において、ソースコード中の改行は、プログラムの実行結果に影響を及ぼさない。これより、プログラムの動作に影響を与えない要素によって、ソースコードの可読性が実現されることがわかる。

1.2　ソフトウェア保守と可読性の関係

ソフトウェアは、ソフトウェアの開発が企画されてから運用が終了するまでを、いくつかの工程に分けることができる。この工程全体をソフトウェアライフサイクルという。ソフトウェアライフサイクルは、「企画」、「要件定義」、「開発」、「運用」、「保守」という工程に分けることができる。そして、ソフトウェアの保守作業は、ソフトウェアライフサイクルのコストの大半を占める。BohemとBasiliは、調査によって、ソフトウェアの保守作業がソフトウェアライフサイクルのコストのうちおよそ70%を占めることを明らかにした[2]。

ソフトウェアの保守作業とはバグの修正や、プログラム外部的な振る舞いを保ちつつ、内部的構造を変化させる一連の作業である、「リファクタリング」[3]などを行う工程である。このように、ソフトウェアの保守作業の多くがプログラムの変更を伴う作業であることから、ソフトウェアの保守作業にかかるコストの多くは、ソフトウェア開発者の人件費によるものであると考えられる。

ソフトウェアの保守作業のしやすさのことを保守性という。保守性が高く保たれているソフトウェアは、保守性が高く保たれていないソフトウェアに比べて、ソフトウェアの保守作業にかかる時間が少なくなると考えることができる。これより、ソフトウェアの保守性を高く保つことで、ソフトウェアの保守コストを削減することができると言える。ソフトウェア

---

1 R. P. L. Buse & W. R. Weimer (2008) p.121

2 B. Boehm & V. R. Basili (2001) p.137

3 M. Fowler, K. Beck, J. Brant, W. Opdyke, d. Roberts (1999) p.9

の保守性に影響を及ぼす要素には、さまざまなものが挙げられる。その要素の一つとして、ソフトウェアのソースコードの品質が考えられる。ソースコードとは、プログラミング言語によって記述された、プログラムを表現するテキストのことである。

ソースコードの品質にはさまざまな指標が存在する。ソースコードの品質は、ソースコードを解析するソフトウェアメトリクスによって計測することができる。ソースコードを解析するソフトウェアメトリクスの代表的なものとして、プログラムの制御フローの複雑さを計測するMeCabeメトリクス(サイクロマティック数)などが挙げられる[4]。

ソースコードの品質に影響を与える要素は他にも存在し、上にあげたもの以外に、ソースコードの可読性が考えられる。一般にソフトウェアの保守作業はソースコードを読解する作業が伴う。ソースコードの可読性が高いほど、ソフトウェア開発者がソースコードの読解に費やす時間を削減することができると考えられる。つまり、可読性の向上によって、ソフトウェアの保守コストの削減が期待できると考えられる。また、可読性は、ソフトウェアの品質に影響を与えることがわかっている。Butler他は、ソースコードの可読性がソフトウェアの品質に影響を与えることを明らかにした[5]。これより、ソースコードの可読性は、ソフトウェア開発において重要な指標であると言える。

1.3　識別子

識別子とは、「プログラム言語における変数名や関数名を表す名前」[6]である。識別子は、関数や、変数のようなプログラムを構成する要素を一位に識別するためのものである。ほとんどのプログラミング言語において、識別子名はソフトウェア開発者が自由に決定することができる。識別子名には単一の単語や、複数単語をつなぎ合わせたものが使用されることがある。一般に、識別子が指すプログラムの構成要素を表現するように識別子を命名することが推奨される。

1.4　識別子名における単語の連結の形式

一般的なプログラミング言語では、識別子名に空白文字(スペース等)を含めることはできない。それゆえ、識別子の命名に複数の単語を用いる場合は何らかの形式に従って単語を連結する必要がある。Table1は複数の単語を連結する際の代表的な形式である。

---

4 阿萬 裕久, 野中 誠, 水野 修, (2011) p.14

5 S. Butler, M.Wermelinger, Y. Yu, H. Sharp, (2009)

6 株式会社オーム社編 (2001) p.288

Table1: 単語の連結方法

形式名	形式	例
アッパーキャメルケース	全ての単語の先頭を大文字にして連結する	UpperCamelCase
ロワーキャメルケース	最初以外の単語の先頭を大文字にして、連結する	lowerCamelCase
アッパースネークケース	全ての文字を大文字にして、アンダースコアで連結する	UPPER_SNAKE_CASE
ロワースネークケース	全ての文字を小文字にして、アンダースコアで連結する	lower_snake_case
アッパーケバブケース	全ての文字を大文字にして、ハイフンで連結する	UPPER-KEBAB-CASE
ロワーケバブケース	全ての文字を小文字にして、ハイフンで連結する	lower-kebab-case

1.5　命名規則

1.2で述べたように、識別子名はソフトウェア開発者が自由に決定することができる。しかし、その自由さが問題となることがある。Sneedはソフトウェア開発者の気まぐれで好きなスポーツ選手などの名前が、識別子につけられることを指摘している[7]。このような問題を防ぐために、プロジェクトによっては「命名規則」が導入されることがある。

命名規則とは、どのように識別子を命名するかを定めたものである。命名規則では識別子名に用いる単語を指定したり、複数単語を連結する際の形式、識別子の文字数、識別子の品詞など様々なものがあり、命名規則によってその指定の内容は異なる。

1.6　識別子の命名と暗黙値

一見、命名規則は、客観的な基準のように思われる。しかし、命名規則は、命名規則を定めたソフトウェア開発者の経験に依存しており、主観的な基準であると考えられる。多くのソフトウェア開発プロジェクトでは、命名規則が導入される。一般に、ソフトウェア開発のプロジェクトには、複数人の開発者が参加することから、ソフトウェア開発のプロジェクトに導入された命名規則は、複数人の開発者に共有されるといえる。世の中にはさまざまな命名規則が存在するが、複数の命名規則の間には共通した規則が存在することがある。以上より命名規則は主観的な基準であるにもかかわらず多くのソフトウェア開発者に支持されており、その命名規則には共通点が存在することから、命名規則の根底には、ソフトウェア開発者の間で共有された暗黙知が存在していると考えることができる。

1.7　java

本項では、javaや、javaの言語機能について述べる。javaは1996年にサンマイクロシステムズ(Sun microsystems)社より提供されたオブジェクト指向プログラミング言語であり、現在はオラクル(Oracle)社の登録商標である。javaは提供から現在までおよそ25年近くの間利用され

7 H. M. Sneed (1996) pp.171-172

ており、歴史が長いプログラミング言語であると言える。Githubにはjavaのオープンソース・ソフトウェアプロジェクトが数多く存在する。これより、javaは提供から現在まで、多くの開発者によって利用されてきたプログラミング言語であると考えることができる。これより、javaには識別子の命名に関するソフトウェア開発者の暗黙値が蓄積されているといえる。これより、識別子名に含まれる単語の品詞間の関係に関する、利用しやすく、有益な法則が得られることが期待される。それゆえ、本研究ではjavaを対象に、調査、分析を行うこととした。

1.7.1　javaの識別子

javaのプログラムは、「Token」、「Comment」、「WhiteSpace」という3つの最小単位によって構成される。さらに、Tokenはプログラムの構文的特性から「Identifier」、「Keyword」、「Separator」、「Operator」の4つに分類することができる[8]。これより、識別子はTokenの一種であり、プログラムの最初単位のであるとわかる。

javaの識別子(Identifier)は、識別子が指すプログラム要素によって分類することができる。本研究では、識別子が指す対象によって「クラス識別子」、「メソッド識別子」、「変数識別子」の3つに識別子を分類する(Table3)。

javaの識別子名には、大文字小文字のアルファベット数字、ドル記号、アンダースコアを使用することができる[9]。これよりjavaは、開発者が識別子名を自由に決定することができるプログラミング言語であると言える。

Table2: javaのTokenの分類

分類	詳細
Identifier	識別子
Keyword	ASCII文字から成る50種類の予約語
Separator	区切り文字 ( ) { } [ ] ; , ... @ ::
Operator	ASCII文字から成る38種類の演算子

8 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.19-20

9 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.22

Table3: 本論文におけるjavaの識別子の分類

分類	識別子が対象とする要素
クラス識別子	クラス
メソッド識別子	メソッド
変数識別子	フィールド変数、ローカル変数

1.7.2　クラス

オブジェクトは、データおよびデータを操作する手続きを、パッケージ化したものである。javaのようなオブジェクト指向(Object Oriented)プログラミング言語では、オブジェクトを扱うが、オブジェクトはクラスをインスタンス化することで生成される。オブジェクトの実装はそのクラスによって定義される。クラスはオブジェクトの内部データとその表現を明記し、オブジェクトが実行するオペレーションを定義する[10]。これより、クラスはオブジェクトを生成するための雛形のようなものだと考えることができる。

javaには通常クラス(NormalClass)と列挙子クラス(EnumClass)が存在する。本研究では通常クラスを「クラス」と呼ぶことにする。また、javaではクラスの宣言に修飾子(Modifier)を含めることができる。修飾子とは、識別子の性質を決める役割を持ったキーワードである。

javaのクラスはDeclaration1のように宣言される[11]。

```
[Modifier] class [クラス名(Identifier)] {
        [クラスの実装]
}

例: Carという名前のクラスを宣言する場合
class Car {
}
```

Declaration1:クラスの宣言

1.7.3　メソッド

メソッド(Method)とは、オブジェクトによってパッケージ化された、データを操作する手続きのことである。メンバー関数とも言われることがある。javaにおいて、メソッドの宣言

10 E. Gammma, R. Helm, R. Johnson, J. Vlissides (2002) pp.24-25

11 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.193-194

はDeclaration2のように行われる[12]。クラスの宣言と同様に、メソッドの宣言にも修飾子を含めることができる。

```
[Modifier] [戻り値の型(Type)] [メソッド名(Identifier)] ([一つ以上の引数]) {
        [メソッドの実装]
}

例: Carクラスに戻り値・引数を持たない"run"というメソッドを宣言する場合
class Car {
        void run() {
        }
}
```

Declaration2:メソッドの宣言

1.7.4　変数

javaの変数には、ローカル変数とフィールド宣言が存在する。一般にローカル変数はメソッド内やforブロック内など、短いスコープで使用される。一方、フィールド変数はクラスのメンバとして宣言される。フィールド変数はクラスのメンバからアクセスされる。また、クラスの外からもアクセスされることがある。したがって、フィールド変数はローカル変数よりも大きなスコープで使用される変数であると言える。ローカル変数の宣言はDeclaration3、フィールド変数の宣言はDeclaration4のように行われる[13]。

```
[変数の型] [ローカル変数名(Identifier)];

例: 文字列を格納する"text"という名前のローカル変数を宣言する場合
String text;
```

Declaration3:ローカル変数の宣言

12 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.227-228

13 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.213-214, 414-415,

```
[Modifier] [変数の型] [フィールド変数名[Identifier];

例: Carクラスに文字列を格納する"text"というフィールド変数を宣言する場合
class Car {
        String text;
}
```

Declaration4:フィールド変数の宣言

1.7.5　アクセス修飾子

javaには「スコープ(Scope)」が存在する。スコープとは、ある識別子を参照できる範囲のことである。javaではクラス、メソッド、フィールド変数の宣言にアクセス修飾子を含めることができる。識別子の宣言にアクセス修飾子を含めることで、スコープを指定することができる。javaのアクセス修飾子は3種類存在する[14]。

Table4: javaのアクセス修飾子

アクセス修飾子	詳細
public	全てのクラスからアクセスが可能
private	現在のクラスのみからアクセスが可能
protected	現在のクラスとサブクラスからのみアクセスが可能
なし(標準)	現在のパッケージからのみアクセスが可能

1.8　javaで推奨される識別子の命名

javaでは、開発者が識別子名を自由に決定することができるが、1.5で述べたように、この自由さが問題となることがある。このような問題を防ぐために命名規則が定められることがある。javaはOracle社によって提供されているプログラミング言語だが、オラクル(Oralcle)社が推奨する命名規則が存在する[15]。この命名規則では、単語の連結の形式と品詞について述べられている。Table5は識別子の種類と単語の連結の形式と品詞についてまとめた表である。

---

14 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.193, 217, 233

15 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.136-139

Table5: javaで推奨される命名規則

識別子の種類	単語の連結の形式	品詞
クラス識別子	アッパーキャメルケース	名詞、名詞句
メソッド識別子	ロワーキャメルケース	動詞、動詞句
変数識別子	ロワーキャメルケース	名詞、名詞句

1.9　javaにおける命名の慣習

変数の中には定数と呼ばれるものが存在する。定数とは、プログラム中で不変な変数である。定数は特別な変数であるといえる。しばしば定数名はアッパースネークケースによって表現されることがある。しかし、この定数名における単語の連結の形式は、定数名における品詞の関係に影響を及ぼさないと考えられるため、本研究では定数における単語の連結の形式を考慮しないものとする。

javaでは、フィールド変数にアクセスするメソッド名が、get~やset~のように命名されることがある。考察では、この点について考慮する必要があると考えられる。

1.10　識別子と可読性の関係

識別子と可読性の関係について、様々な研究が行われている。例えば、Hofmeister他によると、識別子名に単語を使用した場合、文字を使用した場合に比べてソースコードの理解速度が有意に速くなることを明らかにした[16]。これより、識別子によって可読性を向上させることが可能であることがわかる。

ソースコードは、その大半が識別子によって構成されている。DeissenboeckとPizkaは調査によって、ecliplse3.1.1 のソースコードの全文字数のうち71.5%が識別子によるものであることを明らかにした[17]。Source1はjunit4のソースコード[18]の一部を抜粋したものである。このソースコードはjavaによって記述されている。Table6はSouce1中に出現する全て識別子を分類した表である。Source1は14行のソースコードだが、6個の識別子が含まれており、識別子がソースコードの多くを占めていることがわかる。これより、識別子が可読性に大きな影響を持っていると考えられる。

---

16 J. Hofmeister, J. Siegmund, D. V. Holt (2019)
[https://link.springer.com/article/10.1007/s10664-018-9621-x#citeas] (2021/01/24 閲覧)

17 F. Deissenboeck, M. Pizka (2006) pp.261-262

18 Github, junit4/Assert.java at main · junit-team/junit4
[https://github.com/junit-team/junit4/blob/main/src/main/java/junit/framework/Assert.java]
(2021/1/4 16:51閲覧)

```
public class Assert {
    /**
     * Protect constructor since it is a static only class
     */
    protected Assert() {
    }

    /**
     * Asserts that a condition is true. If it isn't it throws
     * an AssertionFailedError with the given message.
     */
    static public void assertTrue(String message, boolean condition) {
        if (!condition) {
            fail(message);
        }
```

Source1: junit4から抜粋したソースコード

Table6: Source1中に出現する識別子

識別子の種類	識別子
クラス識別子	Assert
メソッド識別子	assertTrue, fail
変数識別子	message, condition
その他の識別子	Assert

1.11　可読性と品詞の関係

品詞(Part-Of-Speech)は、自然言語における語の文法的な分類である。そして、品詞間には文法的な関係が存在する。

しばしば、命名規則では品詞に関する規則が定められることがある。1.6ではjavaにおける命名規則の中に品詞があることを確認した。プログラミング言語は、自然言語とは異なる構文を持った言語である。

命名規則がソースコードの可読性を向上させる目的で導入されることと、識別子の命名が品詞に基づいて行われることから、ソースコードの可読性を向上させる効果が品詞にあることが、ソフトウェア開発者の間で広く認められていると考えられる。

1.12　オープンソース・ソフトウェアと可読性の関係

オープンソース・ソフトウェア(Open Source ・ Software)とは、誰もが検査、修正、拡張で

きるソフトウェアのことである[19]。一般に、オープンソース・ソフトウェアの開発では、誰もがソフトウェアを検査、修正、拡張できるという特徴から、不特定多数のソフトウェア開発者が関わることが多い。例えば、javaのテストライブラリであるjunitは150人のソフトウェア開発者によってメンテナンスされている[20]。これより、オープンソース・ソフトウェアのソースコードは、不特定多数のソフトウェア開発者に共有されていることがわかる。それゆえ、オープンソース・ソフトウェアではソースコードの可読性が重要であると考えられる。

オープンソース・ソフトウェアの開発では、ソフトウェアの品質を維持するため、ソースコードの変更を統合する前に、変更内容を検証する「レビュー」という開発プロセスが存在する。このレビューは、ソースコードの変更者とは別の、ソフトウェア開発者によって行われる。それゆえ、オープンソース・ソフトウェアではレビューという開発プロセスによってソースコードの可読性が維持されていると考えられる。

---

19 Opensouce.com, What is open source software? | Opensource.com
[https://opensource.com/resources/what-open-source] (2021/1/1 19:14閲覧)

20 Github, junit-team/junit4: A programmer-oriented testing framework for Java.
[https://github.com/junit-team/junit4] (2021/1/4 16:51閲覧)

第二章　調査手法

本章では、調査手法について述べる。識別子中の品詞間の関係を調べるために、javaのオープンソース・ソフトウェアから識別子を収集し、176のプロジェクトを対象に調査を行った。

2.1　調査対象

識別子から品詞間の関係を調べる上で、可読性が高いソースコードから識別子を抽出する必要があった。1.9で述べたように、オープンソース・ソフトウェアのソースコードは可読性が高いことが期待される。それゆえ、数多くのオープンソース・ソフトウェアが集まるGithub上のjavaプロジェクトを調査対象とした。

2.2　調査手法

本研究では、以下の手順で、各プロジェクトから識別子を収集し、識別子中の単語の品詞を取得した。

1. 識別子の抽出
2. 識別子名の分割
3. 品詞の取得
4. 不正なデータの削除

2.3　識別子の抽出

ソースコードから識別子を抽出するにあたって、AST(Abstract Syntax Tree)解析を用いた。ASTとは、javaオブジェクト表現としてjavaソースコードと対話することを可能にしたオブジェクト表現である[21]が、javaのAST解析を行う代表的なライブラリにはeclipse JDT[22]と、JavaParserが挙げられる[23]が、本研究ではJavaParserを用いた。

ASTはソースコードを木構造として表現しており、この木構造はファイル全体を表すルート(root)と、複数のノード(node)を持っている。ルートから木構造をたどることで、ソースコードの任意の場所へとアクセスすることができる。Figure1はJavaParserを用い

---

21 N. Smith, D. Bruggen, F. Tomassetti (2019) p.1

22 Eclipse, Eclipse Java development tools (JDT) | projects.eclipse.org
[https://projects.eclipse.org/projects/eclipse.jdt] (2021/1/6 閲覧)

23 JavaParser, JavaParser - Home
[https://javaparser.org/] (2021/1/6閲覧)

て、Source2に対しAST解析を行った結果を、ツリーダイアグラムとして表現した図(一部省略)である。Figure1を見ると、CompilationUnitをルートに持つ木構造になっていることがわかる。

本研究ではこのAST解析を行うことで、クラス識別子、メソッド識別子、変数識別子を抽出した。

```
package com.github.javaparser;

import java.time.LocalDateTime;

public class TimePrinter {

  public static void main(String args[]){
    System.out.println(LocalDateTime.now());
  }
}
```

Source2: AST解析を行うソースコードの例

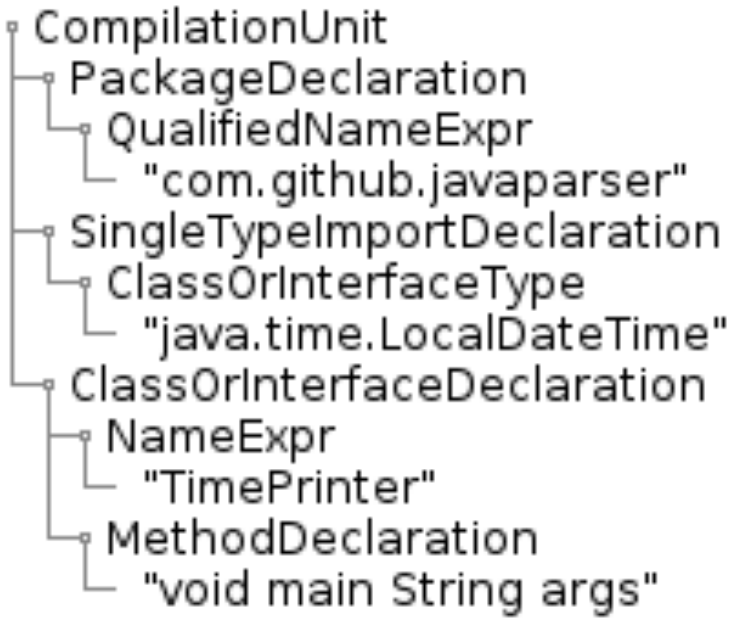

Figure1: Source2のAST解析によって生成された木構造

2.4　識別子名の分割

本研究では品詞の関係を調査するために、識別子から単語を取得する必要がある。通常の英語であれば、単語は空白によって区切られている。しかし、1.3で述べたように、javaの識別子名は複数の単語が連結されている。それゆえ、識別子名を何らかの方法で単語に分割する必要がある。

javaには識別子の種類によって推奨される単語の連結の形式が存在する[24]。そこで、本研究では正規表現を用いて単語を分割した。Source3は本研究で、識別子中の単語を分割する際に使用した正規表現である。また、javaでは識別子名にアンダースコアを使用可能であることから、アッパースネークケース・ロワースネークケースの識別子に対応ができるよう、アンダースコアが含まれる識別子名はアンダースコアで、識別子を分割した。

```
([a-z]+)([A-Z][a-z]+)|([A-Z][a-z]+)
```

Source3: 識別子中の単語の分割に使用した正規表現

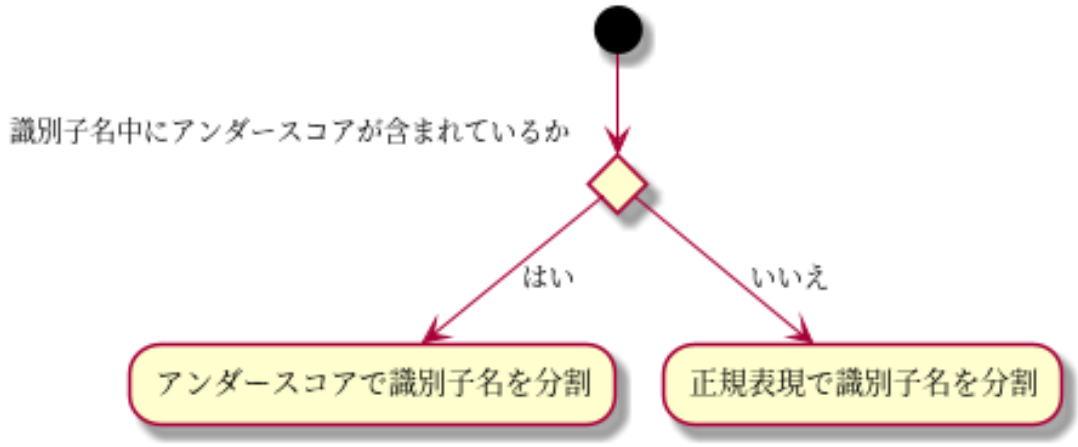

Figure2: 識別子名分割のフローチャート

2.5　品詞の取得

識別子に含まれる単語の品詞の関係を調査する上で、識別子に含まれる単語の品詞を取得する必要がある。そこで、本研究ではpythonのNLTK(Natural Language Toolkit)を用いて品詞を取得した。NLTKは、自然言語を扱うための様々な関数が用意された、Pythonのモジュールである[25]。NLTKで品詞を取得する際、タグセットを指定することができる。今回はNLTK

24 J. Gosling & B. Joy & G. Steele & G. Bracha & A. Buckley (2015) pp.136-139

25 nltk, Natural Language Toolkit — NLTK 3.5 documentation
[https://www.nltk.org/] (2021/1/3 21:19 閲覧)

の品詞を取得する関数で標準とされている、ペンツリーバンク(PennTreebank)のタグセットを用いた[26]。Table7はペンツリーバンクのタグセットの記号とその詳細である。

プログラムによる品詞の取得において、誤った結果が出力される場合がある。重藤他(2012)[27]の研究によると、Penn Treebankに出現した単語から構成される辞書による品詞のタグ付けの精度は97.54％であった。これより、取得した品詞は信頼できるものであると考えられる。

2.6 不正なデータの削除

本調査ではプログラムを用いて識別子の抽出や、識別子に含まれる単語の分割、各単語の品詞の取得を行った。調査に用いたプログラムが信頼できるものか検証するために、各メソッドに対し、単体テストや、結合テストを行い、期待される出力が得られるか検証を行った。そして、JavaParserによる識別子の抽出の段階で、稀に誤った出力がされることがわかった。誤った出力の原因は、Javaのソースコードに含まれるコメント文をJavaParserが構文解析をする際に誤って認識してしまうというものであった。実際、抽出したデータにはjavaでコメント文であることを示す「*」や「/」が混入していた。そこで「*」や「/」が含まれる不正なデータの削除を手作業で行った。

---

26 T. Ann, M. Mitchell, S. Beatrice (2009) p.8

27 重藤他 (2012) p.5

Table7:PennTreebank tagset一覧

タグセットの記号	詳細
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

第三章　調査結果と考察

本章では、調査結果と調査結果に対する考察を述べる。

3.1　調査結果

調査によっておよそ500万個の識別子を取得することに成功した。Table8は調査によって得られたデータの概要である。また、Figure3はクラス識別子、メソッド識別子、変数識別子において、品詞の出現数を集計したグラフである。

Table8:抽出した識別子の概要

識別子の種類	抽出した識別子の数(個)	抽出した単語の数(個)
クラス識別子	365505	126757
メソッド識別子	2729915	7351610
変数識別子	2021672	3576170
合計	5117092	12195355

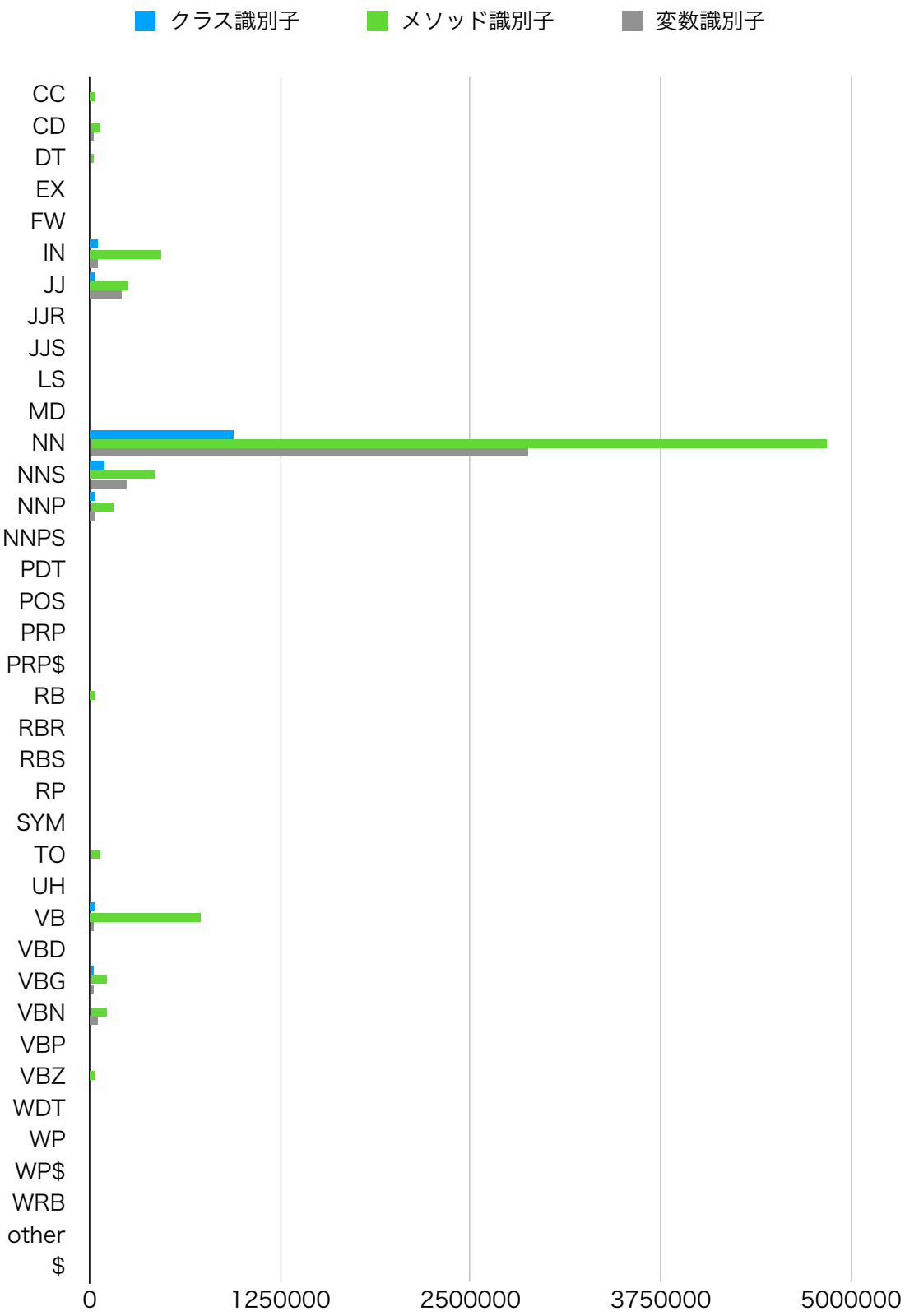

Figure3:各品詞の出現数(個)

3.2　調査結果における偏りの検証

命名規則はソフトウェア開発プロジェクトごとに導入される。そして、各プロジェクトごとに命名規則が異なることがある。それゆえ、調査結果において、偏りが起きる可能性が考えられる。そこで、調査結果のデータの正当性を検証した。

第二章の調査ではGithub上に存在する、176のオープンソース・ソフトウェアのプロジェクトを対象に識別子の抽出を行った。抽出した識別子を、抽出したプロジェクトごとに分類し、算出した割合を算出した。Figure4はここで算出した割合を基に作成したグラフであり、Table9はFigure4の各ラベルの詳細である。

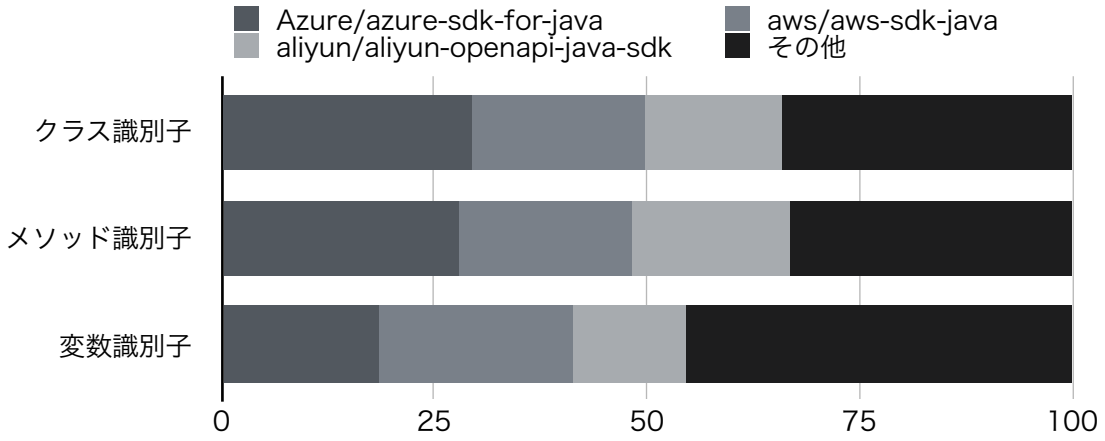

Figure4:抽出元の176プロジェクトにおける識別子の割合(%)

Table9:Figure4の各ラベルの詳細

ラベル名	ラベルの詳細
azure-sdk-for-java	azure-sdk-for-javaから抽出した識別子
aws-sdk-java	aws-sdk-javaから抽出した識別子
aliyun-openapi-java-sdk	aiyun-openapi-java-sdkから抽出した識別子
その他	上記3プロジェクト以外の173プロジェクトから抽出した識別子

Figure4からクラス識別子、メソッド識別子、変数識別子の全てにおいて、「azure-sdk-for-java」、「aws-sdk-java」、「aliyun-openapi-java-sdk」から抽出した識別子の合計が、抽出した識別子全体に対し50%以上を占めていることがわかる。つまり、調査対象

としたプロジェクトの数が176個であったのに対し、抽出した識別子の半数以上が3つのプロジェクトから抽出されたものであった。これより、「azure-sdk-for-java」、「aws-sdk-java」、「aliyun-openapi-java-sdk」によるデータの偏りが存在する可能性が懸念される。そこで本研究では、調査結果におけるデータの偏り検証するために、抽出した識別子をその抽出元が「azure-sdk-for-java」、「aws-sdk-java」、「aliyun-openapi-java-sdk」のいずれかであるものと、そうでないものに分けてデータの類似度を算出した。

本研究では識別子名中に含まれる単語の品詞間の関係について考察を行う。それゆえ、各データにおける品詞ごとの出現傾向が類似していれば、データは類似したものであり、調査結果に偏りはないと考えられる。

本研究では、データ間の類似度をコサイン類似度を用いることで、データ間の類似度の調査を行った。品詞の出現数を集計したデータにおいて、各品詞の種類を軸、各品詞の出現数を対応する軸の座標と考えると、品詞の出現数を集計したデータをベクトルと見なすことができる。ここで、品詞の出現数を集計したデータが2つ存在する時、各データを$\vec{a}$と$\vec{b}$、$\vec{a}$と$\vec{b}$が成す角をθとすると、内積の定義より次式を導出することができる。

$$cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

ここで、$\vec{a}$と$\vec{b}$は品詞の出現数のデータより作成したベクトルであることから、マイナス成分が存在しないことがわかる。それゆえ、$cos\theta$がとりうる値の範囲は$(0 \leqq cos\theta \leqq 1)$である。ここで、コサインの性質より$cos\theta$が0に近づくほど$\vec{a}$と$\vec{b}$の成す角$\theta$は大きくなり、$cos\theta$が1に近づくほど$\vec{a}$と$\vec{b}$の成す角$\theta$は0に近づく。これより、$cos\theta$が1に近づくほど2つのデータは類似し、$cos\theta$が1に近づくほど類似しないと言える。

本研究では、クラス識別子、メソッド識別子、変数識別子それぞれの分類に対し以下の手順を行うことでコサイン類似度を算出した。

1. 調査によって得られたデータを、$A = \{x|$「azure-sdk-for-java」, 「aws-sdk-java」, 「aliyun-openapi-java-sdk」のいずれかから抽出した識別子 $\}$、$B = \overline{A}$という2つのグループに分ける。
2. A、Bそれぞれのデータに対し、各品詞の数の集計を行う。
3. Aにおける集計後のデータを$\vec{a}$、Bにおける集計後のデータを$\vec{b}$とし、コサイン類似度を求める。

Table10は前述の手順によって求めたコサイン類似度である。クラス識別子、メソッド識別子、変数識別子どれにおいてもコサイン類似度は0.997以上であった。これより、「azure-sdk-for-java」、「aws-sdk-java」、「aliyun-openapi-java-sdk」の3プロジェクトと

その他173プロジェクトの調査結果に類似度が認められると言える。よって、本研究ではプロジェクトによる調査結果への影響をないものとして扱うこととした。

Table10:コサイン類似度

	クラス識別子	メソッド識別子	変数識別子
コサイン類似度	0.997585	0.997291	0.999483

3.3　識別子に含まれる単語数

調査結果を基に、各識別子において、収集した識別子に含まれる単語の数の分布のグラフを作成した(Figure5)。グラフにおいて、横軸は一つの識別子に含まれる単語の数であり、縦軸は全体における割合である。また、Table11は、それぞれの識別子に含まれる単語数の中央値である。

Figure5より、クラス識別子に比べ、メソッド識別子と変数識別子の最頻値の単語数が少ないことがわかる。また識別子名中に含まれる単語数の中央値は、クラス識別子では3個であり、メソッド識別子と変数識別子では2個であった。これより、クラス識別子はメソッド識別子と変数識別子に比べ、識別子名中に含まれる単語数が多くなる傾向があると考えられる。

また、識別子中に含まれる単語の数が7個以上の識別子は、クラス識別子、メソッド識別子、変数識別子のどれにおいても、全体の5%未満であった。これより、「識別子名に含まれる単語数が6個以下になるように識別子を命名する」という暗黙知が、ソフトウェア開発者の間に存在していると考えることができる。

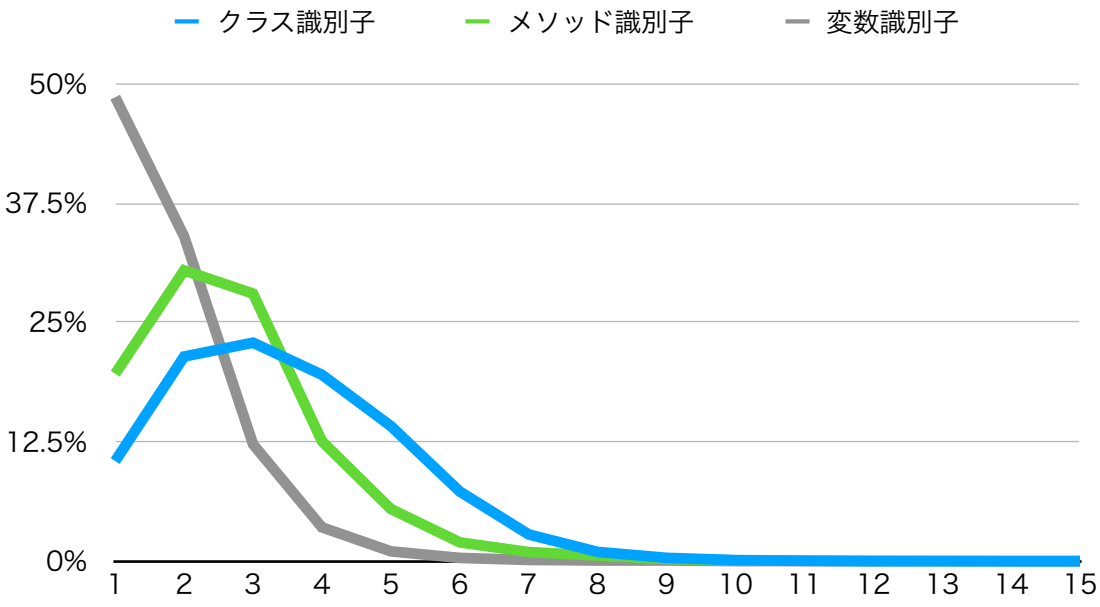

Figure5:各識別子における、識別子中に含まれる単語の分布(単位:個)

Table11:識別子に含まれる単語数の中央値

出現順位	クラス識別子	メソッド識別子	変数識別子
中央値	3	2	2

単位:個

3.4　品詞の出現頻度

1.8において、識別子の分類によって命名規則が異なることを確認した。命名規則が異なれば識別子の分類によって、品詞の出現頻度に差異が生じることが考えられる。本節では、識別子の分類における、品詞の出現頻度について述べる。

3.4.1　全体における品詞の出現頻度

得られたデータから、クラス識別子、メソッド識別子、変数識別子における、各品詞の出現頻度を集計した。そして、集計結果から、出現頻度が上位5つの品詞を表にまとめた(Table12、Table13、Table14)。

これらの表より、クラス識別子、メソッド識別子、変数識別子のどれにおいてもNNの出現頻度が最も高く、65%以上であることが読み取れる。また、どの識別子においてもNN、NNS、JJ、INの出現頻度の順位が上位5番以内であることが読み取れる。さらに、どの識別子においても上位5番以内の品詞の出現頻度の合計が'90%以上であることが読み取れる。

Table12:クラス識別子における品詞の出現頻度

出現順位	品詞の種類	出現頻度(%)
1	NN	74.78
2	NNS	7.59
3	IN	3.74
4	NNP	3.42
5	JJ	3.13
合計		92.66

Table13:メソッド識別子における品詞の出現頻度

出現順位	品詞の種類	出現頻度(%)
1	NN	80.3
2	NNS	6.8
3	JJ	5.7
4	VBN	1.6
5	IN	1.3
合計		95.6

Table14:変数識別子における品詞の出現頻度

出現順位	品詞の種類	出現頻度(%)
1	NN	65.8
2	VB	9.8
3	IN	6.3
4	NNS	5.8
5	JJ	3.5
合計		91.2

3.4.2　識別子に含まれる一番目の単語の品詞の出現頻度

調査によって得られたデータより、クラス識別子、メソッド識別子、変数識別子において、識別子名に含まれる単語のうち、一番目の単語の品詞の種類を集計し、出現頻度が上位5つのデータを表にまとめた(Table15, Table16, Table17)。各識別子においても、上位5つの品詞が、全体の出現頻度のうち90％以上を占めていた。また、各識別子において、NNは最も高い出現頻度であり、その出現頻度は全体の50％以上であった。

メソッド識別子において、VBの出現頻度は、クラス識別子と変数識別子よりも高く、23.6％であった。一方、クラス識別子と変数識別子のNNの出現頻度は、メソッド識別子よりも高かった。

Table15:クラス識別子名の一番目の単語の品詞

出現順位	品詞の種類	出現頻度(%)
1	NN	67.28
2	IN	9.05
3	VB	5.75
4	JJ	4.99
5	NNP	4.23
合計		91.3

Table16:メソッド識別子名の一番目の単語の品詞

出現順位	品詞の種類	出現頻度(%)
1	NN	57.9
2	VB	23.6
3	IN	8.9
4	JJ	2.8
5	NNS	2.6
合計		95.7

Table17:変数識別子の一番目の単語の品詞

出現順位	品詞の種類	出現頻度(%)
1	NN	79.7
2	JJ	8.9
3	NNS	6.0
4	VBN	1.8
5	IN	0.8
合計		97.2

3.4.3　品詞の出現頻度の考察

どの識別子の分類においてもNNの出現頻度が最も多かった。これより、どの識別子においても、識別子名を名詞または名詞句に命名する傾向があると考えられる。

1.6において、javaにおいて、メソッド識別子は動詞または動詞句になるように命名することが推奨されていることを確認した。そこで、この推奨される命名規則が守られているか考察をする。

動詞句は次のように構成される。VPは動詞句を表し、Vは動詞、NPは名詞句を表す。また、→は、左のものが右のものによって構成されることを表現している。

$$VP \rightarrow V\ NP$$

メソッド識別子の単語数の最頻値は2個であった(Table11)。ここで、メソッド識別子中に含まれる単語の数を2個と仮定すると、上の動詞句の構成より、メソッド識別子中における、動詞と動詞以外の品詞の割合は1：1となると考えられる。javaにおいて推奨されている、メソッド識別子の命名規則が守られている場合、メソッド識別子における動詞(VB, VBD, VBG, VBZ, VBN, VBP)の出現頻度は、メソッド識別子に含まれる単語全体の50％程度であると考えられる。ここで、実際にメソッド識別子名中に含まれる単語の数は、常に2個ではないことに留意する必要がある。

調査結果によると、メソッド識別子における動詞の出現頻度は13.3％であった。メソッド識別子が常に2個の単語によって高施されていないことを考慮しても、動詞の出現頻度は予想をはるかに下回っており、メソッド識別子におけるjavaで推奨された命名規則は守られているとは言えないと考えられる。

次に、クラス識別子と変数識別子に対して、メソッド識別子が動詞または動詞句になるように命名される傾向があるか考察する。各識別子の分類における、品詞の出現頻度全体に対する動詞の出現頻度はクラス識別子においては5.2％であり、変数識別子において3.2％であった。さらに、識別子中の1番目の単語の品詞において、メソッド識別子ではVBの出現頻度が23.6％であり、クラス識別子、変数識別子よりも高かった。これよりメソッド識別子は動詞または動詞句になるように命名される傾向があると考えられる。さらに、メソッド識別子における動詞の出現頻度はpublicメソッドでは13.6％であったのに対し、privateメソッドでは11.3％であった。これより、publicメソッドにおいて、メソッド識別子を動詞または動詞句になるように命名する傾向が強いと考えることができる。javaの構文的特性より、publicメソッドはprivateメソッドに比べスコープが大きいことから、スコープが大きいほど、メソッド識別子を動詞または動詞句になるように命名する傾向がソフトウェア開発者の間に存在すると言える。

以上より、ソフトウェア開発者はメソッド識別子を動詞または動詞句になるように命名する傾向があり、その傾向は識別子のスコープが大きくなるほど強くなると言える。

3.5　ある品詞が次にとる品詞の確率

識別子には、複数の単語が含まれる場合が存在する。そこで、識別子中の品詞間の関係を明らかにするために、連続した2つの単語の品詞の関係に着目し、ある品詞が次に

とる品詞の確率を調査した。
本稿では、ある単語の品詞と、その単語が次にとる単語の品詞を、「→」を用いて表現する。例えばJJという品詞が次にとる品詞がNNであった場合、「JJ→NN」となる。また、次にとる品詞が存在しないことを「end」という記号を用いて表現する。例えば、「getImageUrl」という名前の識別子を考える。この識別子は「get、image、url」の3つの単語によって構成されている。それぞれの単語の品詞は、getはVB、imageはNN(1)、urlはNN(2)であった[28]。このとき、VBが次にとる品詞はNN(1)であり、NN(1)が次にとる品詞はNN(2)である。そして、NN(2)の次に来る単語は存在しないため、NN(2)はendを次にとる。これより、「getImageUrl」という識別子に含まれる2つの単語間の品詞の関係は、「VB→NN(1)、NN(1)→NN(2)、NN(2)→end」となる。

Figure6、Figure7、Figure8はある品詞が次にとる品詞の確率についてのヒートマップである。横軸はある品詞であり、縦軸はある品詞が次にとる品詞または記号である。例えば、横軸がJJであり、縦軸がNNであるセルは、JJ→NNの確率を色によって表現している。

ここで、自然言語と識別子の命名における品詞の関係を比較するため、自然言語テキストにおけるある品詞が次にとる品詞の確率を調査し、ヒートマップを作成した。調査対象は、「Alice's Adventures in Wonderland」の1〜12章に含まれるテキストである。このテキストは自然言語処理を行う際、しばしば利用されるテキストである。このテキストに以下の手順を行うことで、ある品詞が次にとる品詞の確率を調査した。

1. テキスト中のアルファベットを全て小文字のアルファベットに置換する
2. テキスト中に存在する省略記法を、Table18にしたがって非省略記法に置換する
3. [\n“”—___;,:：()（）]を[ ]に置換する
4. シングルクォーテーションを[ ]に置する
5. ２個以上連続するスペースを[ ]に置換する
6. [.!?]を文の終わりとして、テキストを文ごとに分割する
7. 各文を単語ごとに分割する。
8. ペンツリーバンクのタグセット(Table7)を用いて、nltkにより、各単語の品詞を取得する
9. 文の終わりの単語が次にとる品詞を「end」として、ある品詞が次にとる品詞の確率を集計する

28　ここではNNという品詞が2回出現する。2個のNNを区別する前に、imageの品詞をNN(1)、urlの品詞をNN(2)と表記している。

FIgure10は上記の手順に従って作成したヒートマップである。横軸が基準となる品詞で、縦軸が横軸の品詞が次にとる品詞を表している。

Table18:省略記法と非省略記法の対応

省略記法	非省略記法
'm	am
're	are
don't	do not
doesn't	does not
didn't'	did not
won't	will not
wanna	want to
gonna	going to
gotta	got to
hafta	have to
needa	need to
outta	out of
kinda	kind of
sorta	sort of
lotta	lot of
lemme	let me
gimme	give me
getcha	get you
gotcha	got you
letcha	let you
betcha	bet you
shoulda'	should have
coulda	could have
woulda	would have
musta	must have
mighta	might have
dunno	do not know

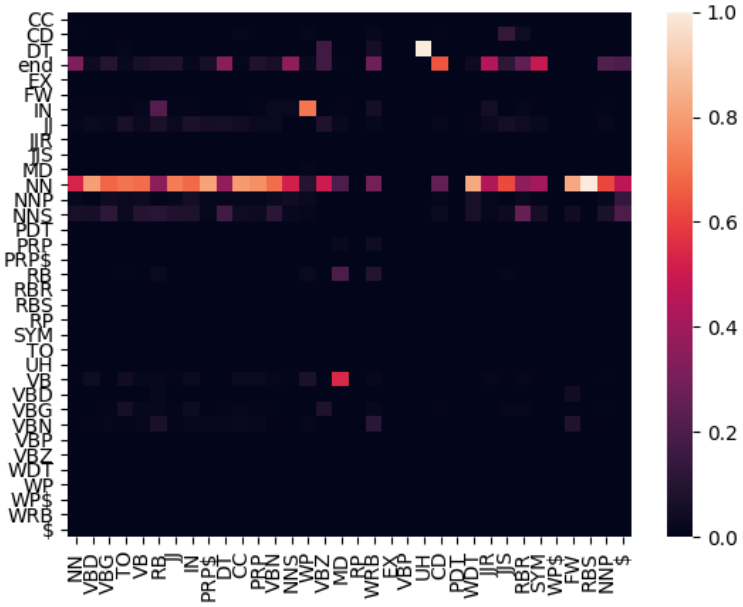

Figure6:クラス識別子における、ある品詞が次にとる品詞の確率

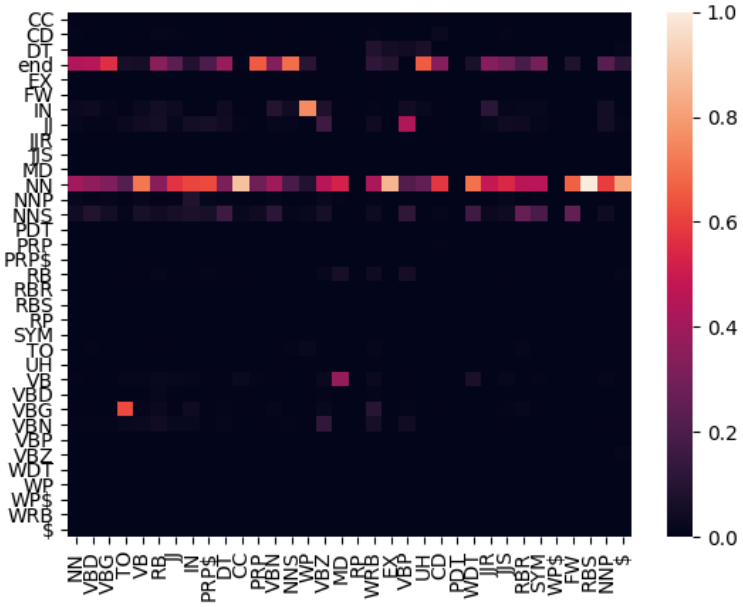

Figure7:メソッド識別子における、ある品詞が次にとる品詞の確率

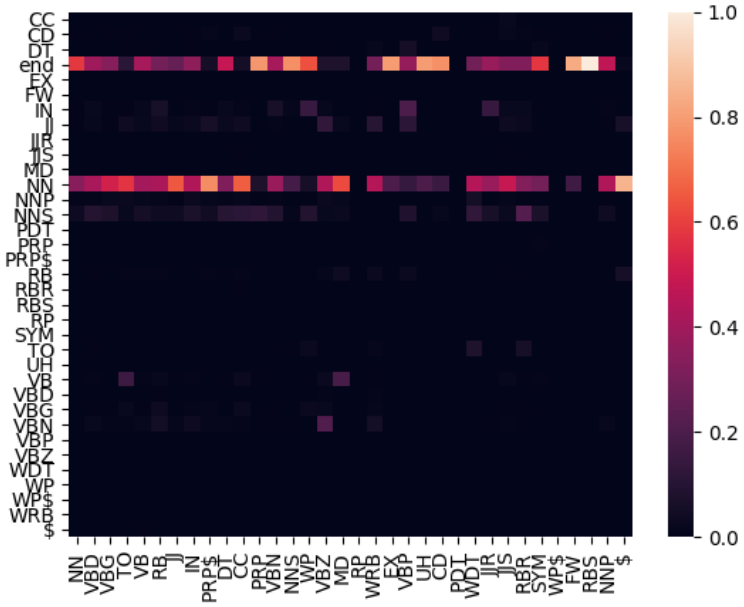

Figure8:変数識別子における、ある品詞が次にとる品詞の確率

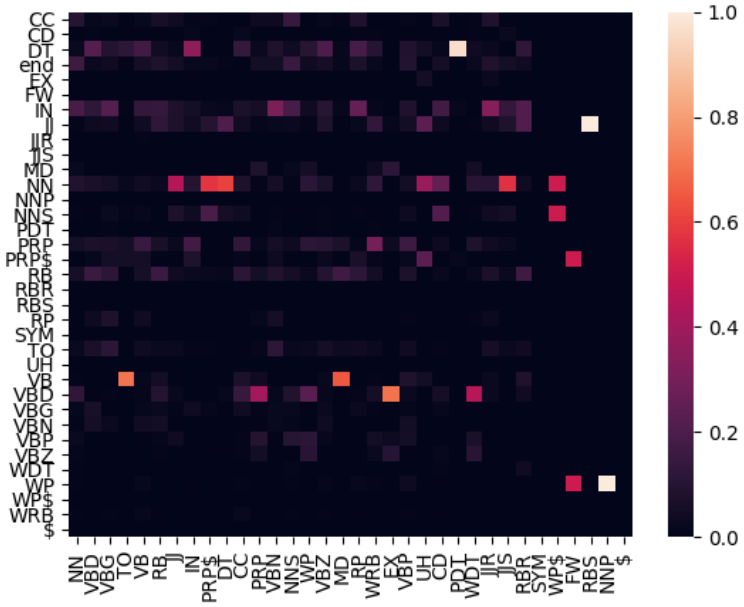

FIgure10:自然言語テキストにおける、ある品詞が次にとる品詞の確率

Figure6、Figure7、Figure8より、クラス識別子、メソッド識別子、変数識別子において、多くの品詞でNNとendを次にとる確率が高いことが読み取れる。これは、取得した品詞のうち、NNが最も多かったことが影響していると考えられる(Figure3)。また、どの識別子の分類においても、MD→VBの確率と、WP→INの確率が共通して高くなっていることが読み取れる。そしてFIgure10においてもMD→VBの確率が高くなっている。このように、自然言語と識別子の命名において、品詞に関わる共通点を、複数確認することができる。

ここで、より客観的に、自然言語と識別子の命名における品詞の関係の共通点を確認するために、カイ2乗検定を用いて、「ある品詞が次にとる品詞の確率」と「自然言語か識別子の命名か」という二つの変数間の「独立性」を検証した。

二つの変数が独立でない場合、ある品詞が次にとる品詞の確率は、自然言語か識別子の命名かによって異なると言える。一方、二つの変数が独立である場合、ある品詞が次にとる品詞の確率は、自然言語か識別子の命名かにかわらず一定である。

したがって、二つの変数が独立である場合、ある品詞が次にとる品詞の確率が自然言語と識別子の命名で共通している可能性があり、二つの変数が独立でない場合、ある品詞が次に取る品詞の確率が自然言語と識別子の命名で共通していないと考えられる。

本研究では、クラス識別子、メソッド識別子、変数識別子の全てのある品詞が次にとる品詞の確率に対して、以下の手順でカイ2乗検定を行った。

1. 自然言語と識別子の命名における、ある品詞が次にとる品詞の確率を基に、Table19のようなクロス集計表を作成する。
2. 有意水準を$\alpha = 0.05$として、帰無仮説H_0と対立仮説H_1を次のように設定する。H_0: ある品詞が次にとる品詞の確率と、自然言語と識別子の命名という二つの変数は独立である。H_1: ある品詞が次にとる品詞の確率と、自然言語と識別子の命名という二つの変数は独立ではない。
3. 1の手順で作成したクロス集計表から、次式により、カイ2乗値(χ^2)を算出する。

k行m列の表において、i行j列の観測度数をO_{ij}、期待度数をE_{ij}、i行の合計をn_i、j列の合計をn_j、全データの合計をnとする。

$$\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{m} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

$$E_{ij} = \frac{n_{i.} n_{.j}}{n}$$

4. 自由度1、$\alpha = 0.05$のとき、$\chi^2_{0.05}(1) = 3.48$より、$\chi^2 > 3.84$の場合H_0を棄却する。

Table19:クロス集計表の例

	NNが次にとる品詞がNNであった数(個)	NNが次にとる品詞がNN以外であった数(個)	合計
自然言語	100	50	150
識別子の命名	200	400	600
合計	300	450	750

単位: 個 (数字は全てダミーデータである)

手順によって得られた結果より、自然言語と識別子名における、ある品詞が次にとる品詞の確率がともに0.01以上であり、H_0を棄却できなかったものを1、そうでないものを0としてヒートマップを作成した(Figure10)。横軸は基準となる品詞を表し、縦軸は基準となる品詞が次にとる品詞を表している。

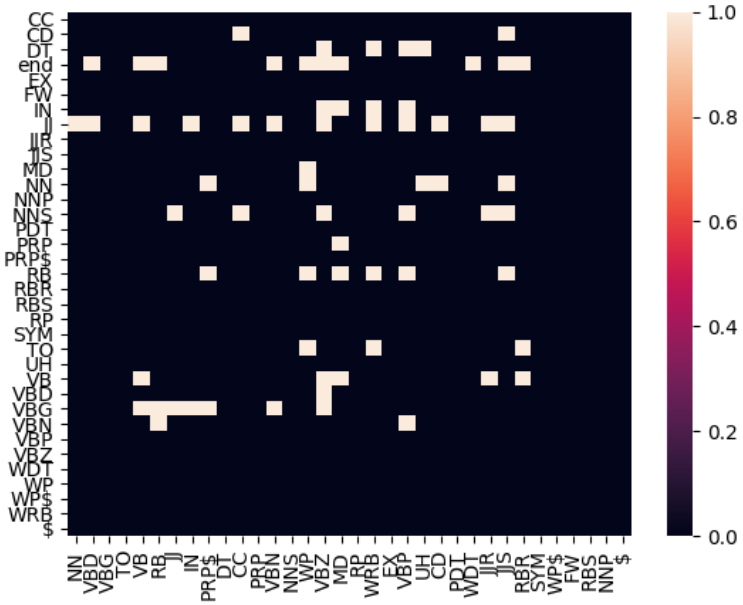

Figure10:カイ2乗検定の結果

FIgure10ではJJ→NNSとJJ→VBGがハイライトされている。JJは形容詞、NNSは複数形の名詞、VBGは~ing形の動詞である。まずJJ→NNSについて、自然言語において形容詞は名詞を修飾する役割があり、この役割が自然言語と識別子の命名に共通していると考えられる。次にJJ→VBGだが、形容詞には動詞を修飾する役割はないため、一見、文法上おかしなように見える。しかし、自然言語において~ing形の動詞には、名詞的用方が存在するため、JJ→VBGは、形容詞が名詞を修飾していると考えられ、自然言語の文法上正しいものであると考えることができる。また、FIgure10ではNN→JJがハイライトされていた。自然言語では、形容詞は名詞を後置修飾することがある。以上より、形容詞の役割が自然言語と識別子の命名に共通していると考えられる。

WRB、RB、RBRはどれも副詞である。自然言語において、副詞は動詞や形容詞を修飾する役割がある。FIgure10ではRB→VBG、RB→VBN、WRB→JJ、RBR→VBがハイライトされていた。これより、副詞の動詞や形容詞を修飾する役割は、自然言語と識別子の命名に共通していると考えられる。さらにFIgure10より、RB→endとRBR→endがハイライトされていることがわかる。自然言語において時や場所を表す副詞は文末が定位置である[29]。これより、時や場所を表す副詞の用法は、自然言語と識別子の命名に共通していると考えられる。

$PRPはmyやhisのような所有格である。自然言語において、所有格は名詞または名詞句を修飾する。FIgure10では$PRP→NN、$PRP→RB、$RBR→VBGがハイライトされていた。自然言語において~ing形の動詞には名詞的用法が存在する。また、名詞句において、名詞が形容詞によって前置修飾される際、その形容詞が副詞によって前置修飾されることがある。これより、$PRPの用法は自然言語と識別子の命名に共通していると考えられる。

FIgure10ではCC→NNがハイライトされていた。CCは等位接続詞であり、andやorである。等位接続詞は文と文を接続したり、名詞と名詞と接続する働きがある。調査結果より、ほとんどの識別子名が6個以下の単語によって構成されることから、識別子名中において、節と節をつなぐ等位接続詞が出現することは極めて稀であると考えられる。これより、自然言語と識別子の命名において、名詞と名詞を接続する役割が共通していると考えられる。

3.6　命名方法の提案

考察より、全体的に識別子名が名詞または名詞句になるように命名される傾向があることがわかった。メソッド識別子は他の分類と比較して、識別子名を動詞または動詞句になるように命名する傾向があることがわかった。また、形容詞、副詞は、所有格は、自然言語の構文と同様の品詞の関係が、識別子の命名においてもあることがわかった。さらに等位接続詞の名詞または名詞句動詞を接続する役割があることがわかった。これらを踏まえ、本稿では以下の命名方法を提案する。

---

29 大西 泰斗, ポーク・マクベイ (2011) pp.250-251

1. 識別子は名詞または名詞句になるように識別子を命名する。
2. メソッド識別子は、可能な場合、動詞または動詞句になるように命名する。
3. 形容詞を用いる場合は自然言語における形容詞の用法に従う。
4. 副詞を用いる場合は自然言語における副詞の用法に従う。
5. 所有格を用いる場合は自然言語における所有格の用法に従う。
6. 等位接続詞を用いる場合は、名詞句と名詞句を接続するように命名する。

提案した命名方法では「自然言語における用法に従う」という文言を採用した。これは、命名方法をより簡素なものにして、使いやすくする狙いがある。仮に、品詞の詳細な使い方に言及した場合、命名方法は複雑なものになる。複雑な命名方法は、ソフトウェア開発者にとって使いづらいだけではなく、ソフトウェアの開発や保守のコストを増大させてしまう可能性がある。それゆえ、シンプルな表現を用いた。

提案した命名方法のうち、1、2は識別子名の一貫性を実現し、4、5、6は識別子名中の単語における、修飾、非修飾における曖昧性を排除する効果が期待される。これにより、ソースコードの可読性を向上させることができると考えられる。

結論

ソースコードの可読性を向上させるには、識別子の命名規則が不可欠である。ソフトウェア開発のプロジェクトにおいて、しばしば識別子の命名規則が定められるが、抽象的な命名規則もあった。そこで、本研究では、識別子の命名に単語が使用されることに着目し、可読性が高いと考えられるオープンソース・ソフトウェアから、識別子中の品詞の関係を抽出した。そして、得られた品詞の関係を基に、識別子の命名手法の提案を行った。

本研究ではjavaを対象に調査、考察を行った。したがって、他のプログラミング言語にも本研究で提案した命名手法はjavaに限定されたものであり、他のプログラミング言語に応用する場合において、別途検証が必要になることに注意する必要がある。また、本研究ではカイ2乗検定により、調査結果に対する分析を行い、品詞間に存在する関係を明らかにした。しかし、カイ2乗検定の結果において、自然言語と識別子における関係が独立であるという帰無仮説を採用したのではなく、帰無仮説を棄却できなかったという消極的なものであることに留意する必要がある。

本研究の課題としては、提案手法が効果的なものであるかに関する検証が挙げられる。本研究では多くのオープンソース・ソフトウェアを対象として調査を行い、その結果をもとに命名手法の提案を行った。この提案した命名手法は、よく使われている命名方法がいい命名方法であるということを前提としている。それゆえ、javaにとっていい命名方法であるとは言い切ることができない。そこで、javaの標準ライブラリを対象として分析を行ったり、既存のソフトウェアメトリクスを用いて、提案手法がjavaにとっていい命名方法であることを示す必要がある。また、連続している3個以上の単語について調査をすることが挙げられる。本研究では調査をする際、2個の連続した単語の品詞に着目し、ある品詞が次にとる品詞の確率を算出した。しかし、3個以上の連続した単語における品詞間の関係について調査が十分でないと考えられる。3個以上の連続した単語について調査すれば、より正確な品詞間の関係を把握することができ、いい命名手法の提案ができると考えられる。さらに、get~やset~といったjavaにおける命名の慣習が識別子中に含まれる単語の品詞の関係及ぼす影響を検証する必要があると考えられる

今後の展望として、提案した命名手法をソフトウェア開発者が実行できているか確認するための、ソースコード解析ツールの作成が考えられる。ソフトウェア開発プロジェクトでは、規定のコーディングスタイルに従ったソースコードかどうかを判定するために、ソースコード解析ツールを導入することがある。このツールの導入により、ソフトウェア開発者にコーディングスタイルガイドに従うことを強制することができる。命名規則に関するソースコード解析ツールの導入により、より一貫性のある識別子の命名を、ソフトウェア開発者に強制することができる。これにより、更なるソースコードの可読性の向上が期待できる。

謝辞

本研究の全過程を通して、常に適切な御指導および御助言を賜りました静岡大学情報学部情報社会学科厨子光政教授に心より深く感謝致します。

本研究を通して、適切な御指導および御助言を頂きました静岡大学情報学部情報科学科酒井三四郎教授に深く感謝申し上げます。

最後に、本研究にあたって、有意義な御指導、御助言および、評価実験へのご協力を頂きました静岡大学情報学部情報社会学科厨子研究室の皆様に深く感謝致します。

参考文献

阿萬 裕久, 野中 誠, 水野 修, 「ソフトウェアメトリクスとデータ分析の基礎」 『コンピュータ ソフトウェア』 28 巻, 3 号, p. 3_12-3_28, 2011

今井 ひとみ, 「英語の命令分の主語に対する統語論及び語用論上の制限」 『名古屋女子大学紀要』, 1983

大西 泰斗, ポーク・マクベイ, 『一億人の英文法』 株式会社ナガセ, 2011

角 征典(訳), Boswell, D., Foucher, T., 『リーダブルコード–より良いコードを書くためのシンプルで実践的なテクニック』 株式会社オーム社, 2012

株式会社オーム社, 『情報技術用語大辞典』 相磯 秀夫監修, オーム社, 2001

佐々木 唯, 肥後 芳樹, 楠本 真二, 「プログラム文の並べ替えに基づくソースコードの可読性向上の試み」 『情報処理学会論文誌』 55, pp. 939-946, 2014

佐藤 匡正, 「プログラム変数の命名--変数はどのように命名されているか」 『島根大学総合理工学部紀要 シリ-ズA』 島根大学総合理工学部, 31, pp. 243-258, 1997
[https://ci.nii.ac.jp/naid/110006939743/] (2020/11/15 閲覧)

式見 遼, 松浦 佐江子, 「識別子の難読化と命名の収集による命名学習方法の提案」 『第74回全国大会講演論文集』 1, pp. 331-332, 2012

重藤 優太郎, 東 藍, 近藤 修平, 北裏 龍太, 坂口 慶祐, 光瀬 智哉, 久本 空海, 吉本 暁文, Frances Yung, 松本 裕治「英語の複単語表現辞書の構築と品詞タグ付けへの応用」 『研究報告自然言語処理（NL）』7, pp. 1-6, 2012
[https://ci.nii.ac.jp/naid/110009486733/](2020/02/08 閲覧)

難波 英嗣, 「テキスト間の類似度の測定」 『情報の科学と技術』 70 巻, 7 号, pp. 373-375, 2020
[https://doi.org/10.18919/jkg.70.7_373] (2020/12/19 閲覧)

本位田 真一, 吉田 和樹(訳), Gammma, E., Helm, R., Johnson, J., Vlissides, J., 『オブジェクト指向における再利用のためのデザインパターン 改訂版』 ソフトバンクパブリッシング

株式会社, 2002

持橋 大地, 能知 宏, 「無限木構造隠れMarkovモデルによる階層的品詞の教師なし学習」『研究報告音声言語情報処理（SLP）』 12, 2016-SLP-111, pp. 1-11, 2016

Boehm, B., and Basili, V., "Software defect reduction top 10 list." *Computer.* 34(1). pp.135–137. 2001.

Buse, R. P. L., and Weimer, W. R., "A metric for software readability." *Proceedings of the 2008 international symposium on Software testing and analysis*. Association for Computing Machinery. New York. USA. pp. 121–130. 2008.
[https://doi.org/10.1145/1390630.1390647] (2021/1/10 閲覧)

Butler, S.; Wermelinger, M.; Yu, Y., and Sharp, H., "Relating Identifier Naming Flaws and Code Quality: an empirical study" *16th Working Conference on Reverse Engineering*. Lille. France. pp. 13-16. 2009.

Deissenboeck, F., and Pizka, M., "Concise and consistent naming." *Software Qual J*. 14. pp. 261–282. 2006.
[https://doi.org/10.1007/s11219-006-9219-1] (2021/1/10 閲覧)

Fakhoury, S.; Roy, D.; Hassan, A., and Arnaoudova, V., "Improving Source Code Readability: Theory and Practice." *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. Montreal. QC. Canada. pp. 2-12. 2019.

Fowler, M.; Beck, K.; Brant, J.; Opdyke, W., and Roberts, D., *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional. 1999.

Gosling, J.; Joy, B.; Steele, G.; Bracha, G., and Buckley, A.; *The Java Language Specification Java SE 8 Edition*. Oracle. 2015.
[https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf] (2020/12/12 閲覧)

Hofmeister, J.; Siegmund, J., and Holt, D. V., "Shorter identifier names take longer to comprehend." *Empir Software Engineering*. 24. pp. 417–443. 2019.

Lawrie, D.; Morrell, C.; Feild, H., and Binkley, David., "Effective identifier names for comprehension and memory." *Innovations Systems and Software Engineering*. 3. pp. 303–318. 2007.
[https://doi.org/10.1007/s11334-007-0031-2] (2020/11/20 閲覧)

OpenSouce.com. "What is open source software? | opensource.com"
[https://opensource.com/resources/what-open-source] (2021/1/1 19:14 閲覧)

Smith, N.; Bruggen, D., and Tomassetti, F., *JavaParser: Visited Analyse, transform and generate your Java code base*. Learnpub. 2019.
[http://leanpub.com/javaparservisited] (2020/12/11 閲覧)

Sneed, H. M., "Object-oriented COBOL recycling." *Proceedings of WCRE '96: 4rd Working Conference on Reverse Engineering*. Monterey. CA. USA. pp. 169-178. 1996.

Sommerville, I., *Software engineering*. 9th ed. pearson. 2009.

Taylor, A.; Marcus, M., and Santorini, B., "The Penn Treebank: An Overview." Text, Speech and Language Technology. vol 20. pp. 5-22. Springer. Dordrecht. 2003.

付録

1. 調査対象のプロジェクト一覧
   以下は調査対象としたプロジェクトの一覧である。

- Azure/azure-sdk-for-java
- aws/aws-sdk-java
- aliyun/aliyun-openapi-java-sdk
- bytedeco/javacpp-presets
- quarkusio/quarkus
- bcgit/bc-java
- google/guava
- SonarSource/sonar-java
- radcortez/javaee7-angular
- alibaba/fastjson
- bjmashibing/java
- kubernetes-client/java
- ramram43210/Java
- apache/servicecomb-java-chassis
- libgdx/libgdx
- tensorflow/java
- querydsl/querydsl
- grpc/grpc-java
- java-native-access/jna
- mongodb/mongo-java-driver
- javaparser/javaparser
- tronprotocol/java-tron
- skylot/jadx
- watson-developer-cloud/java-sdk
- iluwatar/java-design-patterns
- mybatis/mybatis-3
- shopizer-ecommerce/shopizer
- fabric8io/kubernetes-client
- javaee-samples/javaee7-samples
- jboss-javassist/javassist
- GoogleCloudPlatform/java-docs-samples
- graphql-java/graphql-java
- docker-java/docker-java
- realm/realm-java
- DuGuQiuBai/Java
- gitblit/gitblit
- aliyun/aliyun-oss-java-sdk
- brianway/java-learning
- GoogleContainerTools/jib
- thaycacac/java
- square/retrofit

- zeromq/jeromq
- alibaba/arthas
- msgpack/msgpack-java
- agoncal/agoncal-book-javaee7
- mthli/Java
- OpenFeign/feign
- dcm4che/dcm4che
- javamelody/javamelody
- testcontainers/testcontainers-java
- jfinal/jfinal
- zxing/zxing
- google/gson
- searchbox-io/Jest
- rabbitmq/rabbitmq-java-client
- dromara/soul
- ScottOaks/JavaPerformanceTuning
- scribejava/scribejava
- pubnub/java
- json-iterator/java
- EleTeam/Shop-for-JavaWeb
- phishman3579/java-algorithms-implementation
- appium/java-client
- exercism/java
- DreamCats/JavaBooks
- eclipse/paho.mqtt.java
- java-decompiler/jd-gui
- hub4j/github-api
- JustinSDK/JavaSE6Tutorial
- chenhaoxiang/Java
- googleapis/google-cloud-java
- TheAlgorithms/Java
- Vedenin/useful-java-links
- chanjarster/weixin-java-tools
- byhieg/JavaTutorial
- code4craft/webmagic
- qos-ch/slf4j
- json-path/JsonPath
- apereo/java-cas-client
- RichardWarburton/java-8-lambdas-exercises
- redis/jedis
- in28minutes/JavaInterviewQuestionsAndAnswers
- DiUS/java-faker
- lenve/JavaEETest
- hamcrest/JavaHamcrest
- mrdear/JavaWEB
- structurizr/java
- googlemaps/google-maps-services-java
- TooTallNate/Java-WebSocket
- JeffLi1993/java-core-learning-example
- awangdev/LintCode

- ronmamo/reflections
- java8/Java8InAction
- crossbario/autobahn-java
- prometheus/client_java
- objectbox/objectbox-java
- bytedeco/javacv
- rybalkinsd/atom
- influxdata/influxdb-java
- tobato/FastDFS_Client
- moquette-io/moquette
- oldmanpushcart/greys-anatomy
- echoTheLiar/JavaCodeAcc
- bytedeco/javacpp
- biezhi/learn-java8
- 0opslab/opslabJutil
- egzosn/pay-java-parent
- qiyuangong/leetcode
- ChinaLHR/JavaQuarkBBS
- crossoverJie/JCSprout
- jwtk/jjwt
- natural/java2python
- micromasterandroid/java
- HelloWorld521/Java
- mrniko/netty-socketio
- NanoHttpd/nanohttpd
- google/google-java-format
- shyiko/mysql-binlog-connector-java
- qiyaTech/javaCrawling
- javaee-samples/javaee8-samples
- android-async-http/android-async-http
- onblog/JavaMonitor
- alibaba/p3c
- notnoop/java-apns
- tipsy/javalin
- bottleleung/Java
- winterbe/java8-tutorial
- square/javapoet
- akshitagit/JAVA
- AllAlgorithms/java
- nanchen2251/RxJava2Examples
- yasserg/crawler4j
- auth0/java-jwt
- Blankj/awesome-java-leetcode
- otale/tale
- stleary/JSON-java
- amitshekhariitbhu/RxJava2-Android-Samples
- hmkcode/Java
- kaushikgopal/RxJava-Android-Samples
- AllenDowney/ThinkJavaCode
- happyfish100/fastdfs-client-java

- csxiaoyaojianxian/JavaScriptStudy
- socketio/socket.io-client-java
- Beerkay/JavaMultiThreading
- XU-ZHOU/Java
- shekhargulati/java8-the-missing-tutorial
- gaopu/Java
- WritingMinds/ffmpeg-android-java
- binarywang/weixin-java-mp-demo
- yangfuhai/ASimpleCache
- ipinfo/java
- vdurmont/emoji-java
- KikiLetGo/VirusBroadcast
- joeyajames/Java
- kevinsawicki/http-request
- javagrowing/JGrowing
- VerbalExpressions/JavaVerbalExpressions
- biezhi/30-seconds-of-java8
- in28minutes/JavaWebApplicationStepByStep
- enhorse/java-interview
- shakeelstha/java
- SquareSquash/java
- sendbird/SendBird-JavaScript
- Snailclimb/JavaGuide
- Nirman-Rathod/Java
- CarpenterLee/JavaLambdaInternals
- shekhargulati/strman-java
- nramnad/java
- jenkins-docs/simple-java-maven-app
- MindorksOpenSource/from-java-to-kotlin
- MicrosoftDocs/pipelines-java
- h2pl/Java-Tutorial
- SuperMap/iClient-JavaScript
- 17mon/java
- mercyblitz/jsr
- udacity/Just-Java