卒　業　研　究　報　告　書

令和5年度

研究題目

役割と異なる変数名が可読性に与える影響

指導教員　　上　野　秀　剛　　准教授

氏　　名　　石　村　涼　介

令和6年3月8日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

役割と異なる変数名が可読性に与える影響

上野研究室　石村涼介

可読性とは読みやすさの度合いであり, 特に情報科学の分野においてはソースコードの読みやすさを表す. 可読性はソフトウェアの品質に関わる重要な概念である. ソフトウェア保守作業はソースコードを読んで理解することにもっとも多くの時間を要するため, 可読性が高くより短い時間で正しく理解できるソースコードを書くことが開発コストの削減に繋がる. これまでに多くの研究が可読性について分析を行っているが, 可読性に影響を与える要素は数多く, またそれらが複雑に絡み合っているため, さらなる研究が必要である. 本研究では, ソースコードの可読性を「プログラムの処理内容を速く, 正しく理解できる度合い」と定義したうえで, 役割と異なる変数名がソースコードの可読性に与える影響を被験者実験により調査する. 実験ではJava言語で記述されたソースコードを被験者に読んでもらい, アンケートに答えてもらう. (1)理解時間, (2)理解正否, (3)主観評価を測定し, 変数命名規則によって差があるか検証する. 実験の結果, 変数の役割と整合性のある変数名を付けることで, 可読性が上がることが明らかになった. また, 意味のある変数名を付けることで, 主観的な読みやすさが上がることが明らかになった. したがって, 役割と異なる変数名は可読性に悪影響を及ぼすと考えられる.

目次

1 はじめに

ソフトウェアライフサイクルにおいて，保守作業は約70%を占める[1]．また，ソフトウェア保守作業はソースコードを読んで理解することにもっとも時間を要する[2]ため，可読性が高くより短い時間で正しく理解できるソースコードを書くことが開発コストの削減に繋がる．これまでに多くの研究が可読性について分析を行っている[3]が，可読性に作用する要素は数多く，またそれらが複雑に絡み合っているため，さらなる研究が必要である．本研究では，ソースコードの可読性をプログラムの処理内容を速く，正しく理解できる度合いと定義する．

本研究では可読性に作用する要素の内，識別子に着目する．識別子とは他のものと区別するために付けられた名前や番号のことであり，特にソースコードにおいては変数名やメソッド名のことを指す．識別子が可読性に与える影響は大きい．既存研究[3, 4]では識別子長と可読性の関係や，メソッド名と可読性の関係に注目している．識別子の内，変数名は変数の役割を推測し，変数を用いた処理内容の理解を補助するために重要である．

図1にソースコード片の例を示す．図のソースコードは3つの変数(a, b, c)の値を比較して，中央値を出力するプログラムである．このとき，ソースコード中のif-else文の意味を読み取ることで，中央値を出力することを確認できるが，一方で，最後に出力する値を格納する変数名が”median”であることからも，中央値を出力するプログラムであると推測できる．通常，ソースコードを作成する際には変数名がその役割を示すように注意して命名することが望ましいとされる．そのため，変数名が適切に付けられている場合，変数名が意味することを読み取ることでソースコードを詳しく読まなくても処理内容を推測できる．一方で，図2に示すように中央値を格納する変数にもかかわらず”total”と命名されている場合，正しい推測はできない．”total”という変数名から処理内容を推測した場合，合計を出力するプログラムであると誤解する可能性がある．そのため，ソースコードを詳しく読むまで処理内容が分からない．このように，不適切な変数名は直接的にソフトウェアの不具合を起こすわけではないが，コードの可読性を低下させるため，結果的にソフトウェアの品質低下に繋がる恐れがある[6]．しかし，これまでに変数名と可読性の関係に焦点を当てた研究は少なく，さらなる研究が必要である．

本研究は役割と異なる変数名がソースコードの可読性に与える影響を調査することを目的とする．具体的には，被験者実験によって(1)変数名が適切である場合，(2)変数名と処理内容の間に整合性が無い場合，(3)変数名が意味を持たない場合における可読性を比較する．実験では，変数名を書き換えたJavaコードを被験者に読んでもらい，何をしているプログラムか答えるまでの時間を計測する．また，コードの読みやすさを主観的に6段階で評価してもらう．実験の結果を分

```
int median = 0;
int a = 5, b = 8, c = 2;

if(a > b && a < c || a > c && a < b ){
      median = a;
}else if(b > a && b < c || b > c && b < a ){
      median = b;
}else{
      median = c;
}
System.out.println(median);
```

図1中央値を出力するプログラムの一部

```
int total = 0;
int a = 5, b = 8, c = 2;

if(a > b && a < c || a > c && a < b ){
      total = a;
}else if(b > a && b < c || b > c && b < a ){
      total = b;
}else{
      total = c;
}
System.out.println(total);
```

図2不適切な変数名を用いたプログラム

析することで，変数の命名が不適切な場合に悪影響が生じるか明らかになると考えられる．

以下，2章では関連研究について説明し，3章ではソースコードの可読性について説明する．4章では実験内容について説明し，5章では実験結果と考察，6章では本研究のまとめと今後の発展について説明する．

2 関連研究

2.1 ソースコードの可読性

Buseらはソースコードの可読性と様々なソースコード特性の相関を明らかにするため，コードの断片100個に対して学生120人による可読性評価実験を行った[3]. ソースコード特性として，識別子長の平均値と最大値，1行の長さの平均値と最大値など，25種類の数値が用いられた．実験の結果，識別子の数の多さがもっともソースコードの可読性を低下させることが明らかになった．

佐々木らはプログラム文を並べ替えることによるソースコードの可読性向上を試みた[5]. ソースコードの理解において，変数が定義されてから参照されるまでの距離が離れているほどスクロール量が増えたり変数を記憶しておく時間が延びたりするため，プログラムの動作を理解するためのコストが増大する．佐々木らはプログラムの動作を変えない範囲で文を並べ替え，変数の定義と参照の間の距離を短くする手法を提案した．提案手法を評価する実験の結果，実験対象としたJava言語で書かれたメソッド20個のうち16個において可読性が向上した．

これらの研究はソースコードの可読性に注目している点で本研究と関連があるが，変数名とソースコードの可読性の関係に注目した研究は行われておらず，変数名が可読性に与える影響は明らかにされていない．可読性に影響する要素の中でも，特に変数名は自由度が高く，プログラマが簡単に変更できることから間違った命名が起こりやすい．本研究によって変数名が可読性に与える影響を明らかにすることで，プログラムの品質に与える影響を定量的に示すことが可能となる．

2.2 適切な識別子名の影響

山中らは変数に適切な名前が与えられているか評価するため，多数のJavaプログラムから変数名とその特徴を収集し，変数名の適切さを自動的に評価する手法を提案した[6]. ここでいう変数の特徴とは，その変数が登場する行をソースコードから抽出し，Doc2Vecへ入力した結果得られたベクトルのことである．評価実験では，変数xの特徴ベクトルと類似した特徴ベクトルを10個調べ，これら10個の特徴ベクトルに対応する変数y_1〜y_{10}の名前がxの名前と一致するかどうか検証した．一致した場合はxの命名は適切であり，不一致ならばxの命名は不適切であったと判断できる．その結果，6個中4個のソフトウェアでは47〜60%の変数が適切だと判断された．

峯久らはメソッド名とメソッド内容に整合性があるかどうか判定するため，機械学習技術や自然言語処理技術を活用したメソッド名評価手法を提案した[7]. メソッド名も変数名と同様に処理結果に影響はしないものの，処理内容を適切に反映していない場合は可読性を低下させる．峯久らはメソッド名を隠しておき，メ

ソッド内容からメソッド名を推測するというタスクを考え，機械学習と評価実験を行った．評価実験では，テストデータとしてメソッド名とメソッド内容に整合性がある組を1402個，整合性がない組を1403個用意した．そして，モデルが推測したメソッド名が実際のメソッド名と一致しなかった場合を対象とした正解率，適合率，再現率及びF値を計測した．その結果，従来手法を上回るF値＝0.679で不適切なメソッド名を検出することができた．

これらの研究は，識別子名が適切かどうかに注目している点で本研究と関連があるが，ソースコードの可読性との関係に注目した研究は行われていない．つまり，変数名が適切かどうかによって可読性がどれだけ変化するかに注目した研究ではない．本研究では，変数名を書き換えたコードを被験者に読んでもらい，役割と異なる変数名がソースコードの可読性に与える影響を調べる．

なお，可読性と対になる概念として，難読性がある．難読性は読みづらさの度合いを表す．情報セキュリティの分野では，ソースコードの処理内容を攻撃者に理解されないよう，ソースコードを難読化する手法が研究されている．難読化とは，ソースコードの動作を変えずに可読性を著しく下げることである．具体的には不要な分岐を増やしたり，識別子名を変更したりする．一方で，難読化を解除する手法も研究されており，本節で述べた峯久らの研究[7]では難読化解除手法を用いて適切なメソッド名を判定している．難読化と本研究の関係については，5.4節で述べる．

3 ソースコードの可読性と変数名

ソースコードの理解において，識別子が果たす役割は大きい．2章で述べた通り，Buseらは，識別子長がソースコードの可読性に大きな影響を与えることを示している[3]．また，不適切な変数名は直接的にソフトウェアの不具合を起こすわけではないが，コードの可読性を低下させるため，結果的にソフトウェアの品質低下に繋がる恐れがある[6] ここで，識別子長と可読性の関係は，2章で述べたBuseらの論文で取り上げられている．そのため，本研究では取り扱わない．

変数名は変数の役割を推測し，変数を用いた処理内容の理解を補助するために重要である．プログラム中で利用される変数は，プログラムの処理を進める上で何らかの役割をプログラマによって与えられる．役割の例には，中央値の計算結果を格納する，素数かどうかを格納する，などがある．変数には役割に応じた意味と対応する変数名を付けることができる．例えば，”median”という単語は日本語で中央値という意味を持ち，この変数には中央値が格納されることを示す．”is”という単語は真理値を格納するboolean型の変数であるという意味を持ち，日本語で素数という意味を持つ”Prime”と組合わされることで，この変数(isPrime)に素数かどうかを表すbooleanの値が格納されることを示す．

ソースコードの処理内容を理解しようとする作業者は変数名の意味によって変数の役割を推測できる．変数名から変数の役割を推測できれば，プログラムの処理内容の理解にも役立つ可能性がある．例えば，”median”という変数にif-else文によって値が代入され，その内容が処理の最後に画面出力されている場合，中央値を求めるプログラムだと推測できる．変数名が変数の役割を適切に表している場合，変数名からその役割を推測することでプログラム全体の理解を効率的に進めることができ，可読性を高めると考えられる．一方で，変数に役割と異なる名前が付けられている場合は，変数の役割を正しく推測できない．その結果，変数名はプログラムの処理内容の理解に役立たず，それどころか間違った推測を引き起こす可能性がある．つまり，変数の役割と異なる変数名はプログラムの理解を妨げる，すなわち可読性を低下させると考えられる．

そこで，本研究では不適切な命名によって役割と異なる変数名をつけてしまった場合に，可読性に対してどれだけ影響を与えるのかを明らかにする．

4 実験

4.1 概要

Java言語で記述されたプログラムを被験者に読んでもらい，アンケートに答えてもらう実験を行う．被験者は奈良工業高等専門学校の情報工学科に所属する学生12人である．年齢は19歳から20歳で，全員がJava言語を用いたプログラミングの講義を受講済みである．

実験は被験者と実験者のみが居る静かな部屋で実施する．被験者は，Java言語向けの統合開発環境(IDE)であるEclipseを使って，1タスクにつき1つのJavaソースコードを読む．ここで，Eclipseに搭載されている補助機能は使用してもらうが，プログラムの実行は許可しない．実験は2回に分けて行う．1回目は被験者12人のうち9人が普段授業で使用している奈良工業高等専門学校情報工学実験室のPCを使用し，2回目は被験者3人が奈良工業高等専門学校教員研究室で実験用に用意したノートPCを使用する．被験者はソースコードを読んで内容を理解したら，Microsoft Formsでアンケートに回答する．

4.2 変数命名規則

本研究では3種類の変数命名規則を定める．

- **整合性あり**
  変数の役割に対して，適切な変数名を付けること．例えば，中央値が格納される変数に対して”median”という変数名を付ける．

- **整合性なし**
  変数の役割に対して，不適切な（誤解を与えるような）変数名を付けること．例えば，中央値が格納される変数に対して”total”という変数名を付ける．ただし”banana”や”snow”といった，変数の役割を表さない単語は用いない．変数名と変数の役割の整合性を無くすことで，変数名が間違った推測を引き起こす場合の可読性を調べる．

- **意味なし**
  意味のない文字列を変数名とすること．例えば，中央値が格納される変数に対して”pfhcxi”という変数名を付ける．変数名の意味を無くすことで，変数名から変数の役割を全く推測できない場合の可読性を調べる．

本研究の目的は変数の役割と異なる変数名が可読性に与える影響を明らかにすることである．これに基づいて，以下の2つのRQ(Research Question)を設定する．

```
public class Main {
    static int[] coin = {1, 5, 10, 50, 100, 500};

    static int method1(int remain, int index){
        if(remain == 0){
            return 1;
        }
        else if(remain < 0){
            return 0;
        }
        else if(index < 0){
            return 0;
        }
        else {
            return method1(remain-coin[index], index) + method1(remain, index-1);
        }
    }

    public static void main(String[] args) {
        int numOfPattern;
        int total = 70;

        numOfPattern = method1(total, 3);
        System.out.format("%d¥n", numOfPattern);
    }
}
```

図3本実験におけるソースコードの表示例

- **RQ1: 変数名に意味があるときに可読性が向上するか?**
  RQ1に回答するために,「整合性あり」と「意味なし」を比較し,また「整合性なし」と「意味なし」を比較する.
- **RQ2: 変数名に整合性があるときに可読性が向上するか?**
  RQ2に回答するために,「整合性あり」と「整合性なし」を比較する.

4.3 タスク

被験者は1タスクにつき1つのJavaソースコードを読み,プログラムの内容を理解する.ソースコードはEclipse上に表示し,予約語や変数名を色分けして表示する補助機能は全員が使用する.図3に本実験におけるソースコードの表示例を示す.仮引数は青色,ローカル変数は黄色,クラス変数は水色,変数の型やif, returnなどの予約語はオレンジ色で表示される.プログラムの実行は許可しない.

被験者はソースコードの内容を理解したら以下の2項目からなるアンケートに回答する.

- プログラムの動作
  何をしているプログラムなのか自由記述で回答する.なお,本実験は変数名から処理内容を推測できるか評価する必要があることから,「(a+b)/2を求めるプログラム」といったソースコードをそのまま読み上げるような回答はせず,「中央値を求めるプログラム」のような回答をするよう指示する.

- ソースコードの読みやすさ

  6段階（1がもっとも読みにくい，6がもっとも読みやすい）のリッカート尺度から1つを選択する．実験開始時に読みやすさの定義として「プログラムの処理内容を速く，正しく理解できるかどうか」を説明する．段階を偶数にすることで，5段階評価や7段階評価とは異なり,「どちらでもない」という回答を無くすことができる．

一部のタスクはmainメソッド以外のメソッドを含む．mainを除くメソッドの名前について，名前から処理に関する情報が得られないよう，”method1”や”method2”のように書き換える．また，変数名に使われる英単語の意味が分からないと変数名から処理内容を推測できないため，被験者には変数名に使われている英単語の意味をまとめたリストをA4の紙3枚に片面印刷して配布する．リストはタスク中のみ参照可能であり，タスク開始前やタスクとタスクの間は参照禁止とする．

1タスクあたりの制限時間は4分とし，制限時間以内にアンケートへの回答を送信する．タスク開始から回答が送信されるまでの時間を「理解時間」とする．制限時間までに回答できなかった場合は，プログラムの動作を「未回答」と書き換え，ただちに回答を送信する．

タスクは被験者1人につき12個を与える．表1にタスク一覧を示す. 各タスクは1つのJavaファイルからなり，ファイル名から得られる情報を無くすためにファイル名はMain.javaで統一する．12個のタスクにはそれぞれ3種類の変数命名規則を反映したバージョンがあり，被験者は1つのタスクに対してどれか1つのバージョンを読む．また，本実験ではタスクの難易度としてeasyとdifficultを同数用意する．難易度easyはmainメソッドのみからなり，ネストが浅いことから理解しやすいと思われるソースコードである．難易度difficultはmainメソッド以外も登場し，再帰構造を持つことから理解しづらいと思われるソースコードである．プログラムの難易度によって変数命名規則の与える影響が異なる可能性があるため，難易度を区別して実験を行う. タスクの提示順番は，順序効果を考慮しカウンターバランスを行う. 4.2節で述べた変数命名規則を踏まえると，本研究で被験者に読んでもらうプログラムは表2の6種類に分類される．

被験者に提示するソースコードの例として，タスク3（最大値検索）の変数名を書き換える前後のソースコードを図4に示す.「整合性なし」と「意味なし」で用いるソースコードではすべての変数を4.2節で説明した変数命名規則にしたがって書き換える．変数長が可読性に与える影響を抑えるため，可能な限り変数長が書き換える前後で変化しないように書き換えを行う.

4.4 分析

1人の被験者が1つのタスクを行う毎に以下の3つを計測する．

表1タスク一覧

難易度	タスク番号	仕様	変数命名規則		
			整合性あり	整合性なし	意味なし
easy	1	階乗の計算	○		
				○	
					○
	2	素数判定	○		
				○	
					○
	3	最大値検索	○		
				○	
					○
	4	累乗の計算	○		
				○	
					○
	5	配列反転	○		
				○	
					○
	6	10進数を2進数に変換	○		
				○	
					○
difficult	7	ハノイの塔	○		
				○	
					○
	8	経路数の数え上げ	○		
				○	
					○
	9	順列の列挙	○		
				○	
					○
	10	支払う硬貨の組み合わせ数	○		
				○	
					○
	11	組み合わせ記号の計算	○		
				○	
					○
	12	クイックソート	○		
				○	
					○

```
public class Main {
  public static void main(String args[]) {
    int search[] = {2, 19, 5, 17};
    int max = search[0];

    for(int index = 1; index < search.length; index++) {
      if(search[index] > max) {
        max = search[index];
      }
    }

    System.out.println(max);
  }
}
```

(a)変更前（整合性あり）

```
public class Main {
  public static void main(String args[]) {
    int total[] = {2, 19, 5, 17};
    int min = total[0];

    for(int count = 1; count < total.length; count++) {
      if(total[count] > min) {
        min = total[count];
      }
    }

    System.out.println(min);
  }
}
```

(b)変更後（整合性なし）

```
public class Main {
  public static void main(String args[]) {
    int cdwode[] = {2, 19, 5, 17};
    int jje = cdwode[0];

    for(int zpdfs = 1; zpdfs < cdwode.length; zpdfs++) {
      if(cdwode[zpdfs] > jje) {
        jje = cdwode[zpdfs];
      }
    }

    System.out.println(jje);
  }
}
```

(c)変更後（意味なし）

図 4 被験者に提示するソースコードの例

表2変数命名規則と再帰の有無によるプログラムの分類

		変数命名規則		
		整合性あり	整合性なし	意味なし
難易度	easy	整$_{easy}$	不$_{easy}$	無$_{easy}$
	difficult	整$_{difficult}$	不$_{difficult}$	無$_{difficult}$

- **理解時間**

  可読性の定義における「プログラムを速く理解できるか」を確かめるために測定する. 被験者がソースコードを表示してから回答が送信されるまでの秒数を用いる. 制限時間の4分以内にアンケートを回答できなかった場合は理解時間を4分とする. 変数命名規則によって理解時間の平均値に差が生じるか分析する. 理解時間が短いほど可読性が高いとみなす.

- **理解正否**

  タスクで提示したソースコードを正しく理解できたか確かめるために測定する. アンケートで被験者が回答したプログラムの動作の自由記述を実験者が読み, 正解か不正解か判断する. 何をしているプログラムなのか理解できていたら正解とし, 理解できていない場合または制限時間以内に回答していない場合は不正解とする. 変数命名規則によって正解率に差が生じるか分析する. 正解率は全回答数のうち正解数が占める割合であり, 正解率が高いほど可読性が高いとみなす.

- **主観評価**

  被験者が主観的に感じた読みやすさを確かめるために測定する. 変数命名規則によって主観評価の中央値に差が生じるか分析する. 値が大きいほど可読性が高いみなす.

5 結果と考察

5.1 理解時間

難易度ごとの理解時間の平均値を表3に示す. 表の各列は左から難易度, データ数, 平均値（単位は秒）, 標準偏差, 平均値の差を検定したときのp値を表す. 理解時間の平均値はdifficultと比べてeasyの方が70.1秒短く, easyの方が可読性が高いといえる. また, difficultの理解時間は236.4秒と制限時間の4分（240秒）に近く, 標準偏差も小さいことから, 制限時間ぎりぎりまでかかったケースや間に合わなかったケースが多いといえる. Welchのt検定を行った結果, 異なる難易度間で理解時間の平均値に有意差が見られた($p < 0.001$). 以上より, 難易度の設定は適切であったと考えられる.

変数命名規則ごとの理解時間の平均を表4に示す. 理解時間の平均値は「整合性あり」がもっとも小さく,「整合性なし」がもっとも大きい.「意味なし」は両者のほとんど中間であった. よって, 可読性は「整合性あり」がもっとも高く, 次いで「意味なし」,「整合性なし」であるといえる. また, 標準偏差に大きな差がないため, データのばらつき度合いに大きな差はないといえる. one-way ANOVAの結果, 異なる変数命名規則の間で理解時間の平均値に有意差は見られなかった($p=0.187$).

6つの群ごとの理解時間の平均を表5に示す. まず, 同表上部に示した難易度easyについて述べる. 平均値は小さい方から順に「整合性あり」,「意味なし」,「整合性なし」だった. また, 標準偏差に大きな差がないため, データのばらつき度合いに大きな差はないといえる. one-way ANOVAの結果, 異なる群の間で理解時間の平均値に有意差は見られなかった($p=0.08$). したがって,「整合性あり」がもっとも可読性が高く, 仮説通りの結果である.

次に, 同表下部に示した難易度difficultについて述べる. 平均値は小さい方から順に「意味なし」,「整合性あり」,「整合性なし」だった. また, 標準偏差には20ほどの差が見られ, 難易度で区別する前と比べてばらつきが大きくなっている. one-way ANOVAの結果, 異なる群の間で理解時間の平均値に有意差が見られた($p=0.04$). Tukey法によって多重比較を行った結果,「整合性なし」と「意味なし」の間に有意差が見られた($p=0.05$). したがって,「意味なし」がもっとも可読性が高く, 仮説に反する結果である.

タスクの難易度によって変数命名規則が可読性に与える影響が異なる原因を考

表3 難易度ごとの理解時間

	n	平均値[s]	標準偏差	p値
easy	72	166.3	51.1	$p<0.001$
difficult	72	236.4	22.9	

表 4 変数命名規則ごとの理解時間

	n	平均値 [s]	標準偏差	p 値
整合性あり	48	191.7	58.9	p=0.187
整合性なし	48	211.5	45.0	
意味なし	48	200.8	52.0	

表 5 6つの群ごとの理解時間

難易度	変数命名規則		n	平均値 [s]	標準偏差	p 値
easy	整合性あり	整$_{easy}$	24	147.3	51.0	p=0.08
	整合性なし	不$_{easy}$	24	178.0	41.2	
	意味なし	無$_{easy}$	24	173.5	54.7	
difficult	整合性あり	整$_{difficult}$	24	236.0	20.1	p=0.04
	整合性なし	不$_{difficult}$	24	245.0	9.6	
	意味なし	無$_{difficult}$	24	228.1	30.5	

察する．easyにおいて，可読性の高い方から順に並べると「整合性あり」,「意味なし」,「整合性なし」である．一方，difficultにおいて，可読性の高い方から順に並べると「意味なし」,「整合性あり」,「整合性なし」である．このように，タスクの難易度によって変数命名規則が可読性に与える影響が異なる．ここで，3章で説明した「変数の役割と異なる変数名はプログラムの可読性を低下させるのではないか」という仮説について考える．難易度easyについてはこの仮説通りであるが，難易度difficultについては仮説に反する．原因の1つとして，difficultのタスクが非常に難しいことが挙げられる．5.2節で後述するが，difficultの正解数は72件中6件である．つまり，変数命名規則に関わらず，難易度difficultのタスクは不正解が大半を占める．よって,「整合性あり」の理解時間がdifficultにおいても長くなり，結果的に変数命名規則が可読性に与える影響がeasyと一致しなかったと考えられる．

5.2 理解正否

難易度ごとの理解正否を表6に示す．表の各列は左から難易度，全データ数，正解数，不正解数，正解率，正解率の差を検定したときのp値を表す．正解率は難易度easyの方が70%以上高い．よって，難易度easyの方が可読性が高いといえる．カイ二乗検定を行った結果，異なる難易度間で正解率に有意差が見られた($p < 0.001$)．以上より，難易度の設定は適切であったと考えられる．

変数命名規則ごとの理解正否を表7に示す．正解率は「整合性あり」がもっとも高く,「整合性なし」と「意味なし」は同率である．よって，可読性は「整合性あり」

表 6 難易度ごとの理解正否

	n	正解	不正解	正解率	p 値
easy	72	60	12	83.3%	p<0.001
difficult	72	6	66	8.3%	

表 7 変数命名規則ごとの理解正否

	n	正解	不正解	正解率	p 値
整合性あり	48	28	20	58.3%	p=0.104
整合性なし	48	19	29	39.6%	
意味なし	48	19	29	39.6%	

がもっとも高く,「意味なし」と「整合性なし」は同程度であるといえる. なお, カイ二乗検定を行った結果, 異なる変数命名規則の間で正解率に有意差は見られなかった (p=0.104).

6つの群ごとの理解正否を表8に示す. まず, 同表上部に示した難易度easyについて述べる. 正解率は「整合性あり」がもっとも高く,「整合性なし」と「意味なし」は同率である. よって, 可読性は「整合性あり」がもっとも高く,「意味なし」と「整合性なし」は同程度であるといえる. カイ二乗検定を行った結果, 異なる群の間で正解率に有意差が見られた (p=0.03). ボンフェローニ補正をかけたカイ二乗検定によって多重比較を行った結果,「整合性あり」と「意味なし」の間と,「整合性あり」と「整合性なし」の間に有意差が見られた (p=0.03). したがって,「整合性あり」がもっとも可読性が高く, 仮説通りの結果である.

次に, 同表下部に示した難易度difficultについて述べる. 正解率は「整合性あり」がもっとも高く,「整合性なし」と「意味なし」は同率である. よって, 可読性は「整合性あり」がもっとも高く,「意味なし」と「整合性なし」は同程度であるといえる. カイ二乗検定を行った結果, 異なる群の間で正解率に有意差が見られなかった (p=0.19).

表 8 6つの群ごとの理解正否

難易度	変数命名規則		n	正解	不正解	正解率	p 値
easy	整合性あり	$整_{easy}$	24	24	0	100.0%	p=0.03
	整合性なし	$不_{easy}$	24	18	6	75.0%	
	意味なし	$無_{easy}$	24	18	6	75.0%	
difficult	整合性あり	$整_{difficult}$	24	4	20	16.7%	p=0.19
	整合性なし	$不_{difficult}$	24	1	23	4.2%	
	意味なし	$無_{difficult}$	24	1	23	4.2%	

5.3 主観評価

主観評価は6段階評価であるから，1〜3は読みづらいと判断でき，4〜6は読みやすいと判断できる．難易度ごとの主観評価を表9に示す．表の各列は左から難易度，データ数，中央値，中央値の差を検定したときのp値を表す．難易度easyの中央値は5であるから，読みやすいといえる．一方，難易度difficultの中央値は3であるから，読みづらいといえる．両者を比較すると，難易度easyの方が読みやすいといえる．U検定を行った結果，異なる難易度間で主観評価の中央値に有意差が見られた($p < 0.001$)．以上より，難易度の設定は適切であったと考えられる．

変数命名規則ごとの主観評価を表10に示す．中央値は「整合性あり」が5であるから，読みやすいといえる．一方,「整合性なし」の中央値は3であり,「意味なし」の中央値は2であるから，いずれも読みづらいといえる．よって，読みやすい方から順に並べると「整合性あり」,「整合性なし」,「意味なし」であるといえる．クラスカルウォリス検定を行った結果，異なる変数命名規則の間で読みやすさの中央値に有意差が見られた($p < 0.001$)．また，ボンフェローニ補正をかけたU検定によって多重比較を行った結果,「整合性あり」と「整合性なし」の間と,「整合性あり」と「意味なし」の間には有意差が見られた($p < 0.001$)．

6つの群ごとの主観評価を表11に示す．まず，同表上部に示した難易度easyについて述べる.「整合性あり」の中央値は6であり，もっとも読みやすいといえる．一方,「整合性なし」と「意味なし」の中央値は4であり，読みやすいが「整合性あり」には及ばない．よって，読みやすさは「整合性あり」がもっとも優れており,「意味なし」と「整合性なし」は同程度であるといえる．クラスカルウォリス検定を行った結果，異なる群の間で読みやすさの中央値には有意差が見られた($p < 0.001$)．また，ボンフェローニ補正をかけたU検定によって多重比較を行った結果，整$_{easy}$と不$_{easy}$の間と，整$_{easy}$と無$_{easy}$の間には有意差が見られた($p < 0.001$)．よって，主観評価は「整合性あり」がもっとも優れており，仮説通りの結果である．

次に，同表下部に示した難易度difficultについて述べる.「整合性あり」と「整合性なし」の中央値は3であり，同程度読みづらいといえる.「意味なし」の中央値は1であり，もっとも読みづらいといえる．よって，読みやすさは「意味なし」がもっとも劣っており,「整合性あり」と「整合性なし」は同程度であるといえる．クラスカルウォリス検定を行った結果，異なる群の間で読みやすさの中央値には有意差が見られた($p < 0.001$)．ボンフェローニ補正をかけたU検定によって多重比較を行っ

表9 難易度ごとの主観評価

	n	中央値	p値
easy	72	5	p<0.001
difficult	72	3	

表 10 変数命名規則ごとの主観評価

	n	中央値	p 値
整合性あり	48	5	p<0.001
整合性なし	48	3	
意味なし	48	2	

表 11 6つの群ごとの主観評価

難易度	変数命名規則		n	中央値	p 値
easy	整合性あり	$整_{easy}$	24	6	p<0.001
	整合性なし	$不_{easy}$	24	4	
	意味なし	$無_{easy}$	24	4	
difficult	整合性あり	$整_{difficult}$	24	3	p<0.001
	整合性なし	$不_{difficult}$	24	3	
	意味なし	$無_{difficult}$	24	1	

た結果，$整_{difficult}$ と $無_{difficult}$ の間 (p=0.006) と，$不_{difficult}$ と $無_{difficult}$ の間 (p=0.02) には有意差が見られた. よって，主観評価は「整合性あり」と「整合性なし」が同程度優れており，仮説に部分的に反した結果である．

難易度 difficult において「整合性あり」と「整合性なし」の読みやすさが同程度であった理由を考察する．理由は2つ考えられる．1つ目は，5.1 節で述べたのと同様に，難易度 difficult のタスクが非常に難しいことが挙げられる．1つ目の理由によって，難易度 easy と比較して，各変数命名規則の読みやすさが下がったと考えられる．2つ目は，被験者が「整合性なし」のタスクをある程度読めると感じたからだと思われる．変数名に意味さえあれば，理解正否に関わらず，被験者は「読める」と感じていると予想される．具体的には，難易度 difficult のタスクは難しいため，多くの被験者は時間内に理解すること (コードの流れを詳しく追うこと) を諦め，ソースコード全体を眺め，目に入った変数名を基に主観評価を決定したと予想される．2つ目の理由によって,「意味なし」($無_{difficult}$) と比較して,「整合性なし」の読みやすさが 3 まで上がったと考えられる．

5.4 RQ への解答

- **RQ1: 変数名に意味があるときに可読性が上がるか?**

  ここでは,「整合性あり」と「意味なし」を比較し，また「整合性なし」と「意味なし」を比較する．まず,「整合性あり」と「意味なし」を比較する．5.1 節で述べた通り，(1) 難易度を区別しない，(2) 難易度 easy の場合において,「整合性あり」は「意味なし」より理解時間が短かった．(3) 難易度 difficult の場合

は「意味なし」の方が理解時間が短かったが，その理由は5.1節で考察済である．5.2節で述べた通り，全ての場合において,「整合性あり」は「意味なし」より正答率が高かった．5.3節で述べた通り，全ての場合において,「整合性あり」は「意味なし」より主観評価が優れていた．

次に,「整合性なし」と「意味なし」を比較する．5.1節で述べた通り，全ての場合において,「意味なし」は「整合性なし」より理解時間が短かった．5.2節で述べた通り，全ての場合において,「意味なし」と「整合性なし」の正答率は同じであった．5.3節で述べた通り，全ての場合において,「整合性なし」は「意味なし」以上の主観評価を記録した．

したがって，RQ1への回答としては,「意味のある変数名を付けることで主観的な読みやすさは上がるが，本研究で定義した可読性は必ずしも上がらない」となる．ここで「本研究で定義した可読性」とは，1章で定義した「プログラムの内容を速く，正しく理解できるかどうか」を指す．つまり，理解時間と正解率によって表現される，ソースコードの性質のことである．

なお,「意味なし」の変数名は，意味がある場合と比べて，変数の区別や追跡にかかるコストが大きい可能性がある．例えば，変数”median”を区別，追跡するコストよりも変数”pfhcxi”を区別，追跡するコストの方が大きい可能性がある．”median”の6文字にはまとまりがあるが，”pfhcxi”の6文字にはまとまりがないため，記憶コストが増大し，区別や追跡が困難になると考えられる．このように本研究では意図的に役割と異なる変数名をつけたため，一種の難読化を行ったといえる．

- **RQ2: 変数名に整合性があるときに可読性が上がるか?**

  ここでは,「整合性あり」と「整合性なし」を比較する．5.1節で述べた通り，全ての場合において,「整合性あり」は「整合性なし」より理解時間が短かった．5.2節で述べた通り，全ての場合において,「整合性あり」は「整合性なし」より正答率が高かった．5.3節で述べた通り，(1)難易度を区別しない，(2)難易度easyの場合において,「整合性あり」は「整合性なし」より主観評価が優れていた．(3)難易度difficultの場合は両者の主観評価が同程度であったが，その理由は5.3節で考察済である．

  したがって，RQ2への回答としては,「整合性のある変数名を付けることで，本研究で定義した可読性は上がるが，難しいプログラムでは主観的な読みやすさは上がらない」となる．

5.5 妥当性への脅威

5.5.1 被験者の英語習熟度について

被験者にはタスクに登場する英単語をまとめたリストを配布し，分からない単語があれば参照するよう指示した．ここで，英語習熟度の高い被験者はリストを参照する必要がないが，英語習熟度の低い被験者は逐一リストを参照する必要がある．そのため，リストを参照している時間が被験者によって異なり，実験結果に影響した可能性がある．

5.5.2 被験者のソースコード読解力について

被験者全員の年齢は近く，全員がJava言語の講義を受講済みであるが，ソースコード読解力には個人差があると考えられる．したがって，Java言語の使用経験に関するアンケートを実施すればソースコードの読解力(初級者，上級者など)を考慮した分析を行える．

5.5.3 「意味なし」と「整合性なし」の理解時間について

実験設定を変化させた場合に関して考察する．まず，制限時間を無くした場合，難しいタスクはいつまでも理解できないため，理解時間により大きな差が出ると予想される．また，被験者は理解できるまで読み続けるため，正解率は全体的に上がって差が小さくなる可能性がある．次に，プログラムの行数を増やした場合，「意味なし」の変数の追跡に時間がかかり，「整合性なし」より理解時間が長くなる可能性がある．最後に，区別しにくい変数名(”byoivx”と”byqivx”など)を多用した場合，同様に「意味なし」の変数を追跡しにくくなり，「整合性なし」より理解時間が長くなる可能性がある．一方「整合性なし」で区別しにくい変数名を多用しても，意味で区別できるため，時間はそれほど長くならないと予想される(”minimum”と”maximum”など)．つまり，実験設定次第で「意味なし」と「整合性なし」の可読性は逆転する可能性がある．

6 おわりに

本研究では，役割と異なる変数名がソースコードの可読性に与える影響を調べた．具体的には，被験者実験によって(1)変数名が適切である場合，(2)変数名と処理内容の間に整合性が無い場合，(3)変数名が意味を持たない場合における可読性を比較した．実験では被験者にJavaコードを読んでもらい，理解にかかった時間，プログラム理解の正否，主観評価による読みやすさを記録した．

実験の結果，変数の役割と整合性のある変数名を付けることで，本研究で定義した可読性が上がることが明らかになった．また，意味のある変数名を付けることで，主観的な読みやすさが上がることが明らかになった．したがって，役割と異なる変数名は可読性に悪影響を及ぼすと考えられる．

今後の展望としては，より大規模なコードを長時間かけて読んでもらうことが考えられる．本研究で用いたソースコードはいずれも100行以内の小規模なものであった．また，制限時間は4分であり，実際のソフトウェア保守作業に比べると，コードの規模も時間の長さも不足していたと推察される．長時間の作業では作業者の疲れが実験結果に与える影響が大きくなると考えられる．なぜなら，役割と異なる変数名は主観的な読みやすさを低下させ，作業者のストレスの原因になるからである．このようなストレスが長時間かかった際に，理解時間や理解正否に現れる影響は，本研究の結果とは異なる可能性がある．つまり，本研究では見られなかった有意差が生じたり，可読性の順位が入れ替わったりする可能性がある．

謝辞

本研究を進めるにあたり, 多くの方々にご指導, ご協力いただきました. この場を借りてお礼申し上げます.

直接指導いただいた, 指導教員の上野秀剛准教授には, 日頃より実験設定や分析方法など様々な助言をいだたきました. 心より感謝申し上げます.

査読いただいた山口智浩教授には, いくつも貴重なご指摘をいただき, 本研究をよりよいものにすることができました. ありがとうございました.

また, 実験に参加いただいた被験者の方々にも深く御礼申し上げます. ありがとうございました.

参考文献

[1] B. Boehm, V. Basili: ”Software Defect Reduction Top 10 List”, Computer, Vol.34, No.1, pp.135-137 (2001).

[2] A. Ko, B. Myers, M. Coblenz, H. Aung: ”An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks”, IEEE Transactions on Software Engineering, Vol.32, No.12, pp.971-987 (2006)

[3] R. Buse, W. Weimer: ”Learning a Metric for Code Readability”, IEEE Transactions on Software Engineering, Vol.36, No.4, pp.546-558 (2010).

[4] 鬼塚勇弥, 早瀬康裕, 石尾隆, 井上克郎: ”ソースコード中に出現する動詞-目的語関係を利用したメソッド名の命名支援手法”, 電子情報通信学会研究報告, Vol.111, No.481, pp.1-6 (2012).

[5] 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, Vol.55, No.2, pp.939-946 (2014).

[6] 山中啓太, 阿萬裕久, 川原稔: ”プログラムスライスとDoc2Vecを用いた変数名評価法の提案”, コンピュータソフトウェア, Vol.38, No.4, pp.9-15 (2021).

[7] 峯久朋也, 阿萬裕久, 川原稔: ”ソースコードの難読化解除手法を活用したメソッド名の整合性評価”, ソフトウェア工学の基礎, Vol.28, No.9, pp.81-90 (2021).

付録

被験者に配布した英単語リストを表12,13に示す. 英単語リストにおけるタスク1〜3は表1におけるタスク1(階乗の計算)の「整合性あり」「意味なし」「整合性なし」バージョンにそれぞれ対応している. 同様に, 英単語リストにおけるタスク4〜6は表1におけるタスク2(素数判定)の「整合性あり」「意味なし」「整合性なし」バージョンにそれぞれ対応している. つまりタスク番号2,5,8,11,...,35は「意味なし」のタスクであり, 変数名に意味が無いため, 英単語リストに登場しない.

「意味なし」変数名の詳細について述べる. 要素数(全タスクに登場する「意味なし」変数名の合計)は49個である. 変数長の分布を表14に示す. 変数を区別可能な文字数は, 1字が41個, 2字が8個である. ここで, 区別可能とは, そのタスク(ソースコード)内で一意に変数を区別できるということである. よってアルファベットの数(26個)よりも1字で区別可能な変数が多い. 実験設定としては, 全ての変数名を先頭1字で区別できるよう統一すべきだったと考えられる.

被験者が回答したアンケートを図5に示す.

表 12 英単語リスト 1

タスク 1	result	結果
	factorial	階乗
タスク 3	length	長さ
	average	平均値
タスク 4	target	標的
	prime	素数
	each	それぞれ
タスク 6	format	書式，型
	error	誤り
	count	数を数える
タスク 7	search	探索
	max	最大値
	index	添え字
タスク 9	total	合計
	min	最小値
	count	数を数える
タスク 10	base	底（てい）
	exponent	指数
	answer	答え
	count	数を数える
タスク 12	denominator	分母
	numerator	分子
	total	合計
	index	添え字
タスク 13	target	標的
	index	添え字
	temp	一時的な
タスク 15	bottom	基底
	count	数を数える
	length	長さ
タスク 16	decimal	10 進数
	binary	2 進数
タスク 18	right	右
	left	左
タスク 19	left	左
	center	真ん中
	right	右
	disk	円盤

表 13 英単語リスト 2

タスク 21	length	長さ
	input	入力
	temp	一時的な
	output	出力
	index	添え字
タスク 25	permutation	順列
	index	添え字
	length	長さ
	flag	フラグ，旗
タスク 27	select	選ぶ
	count	数を数える
	left	左
	center	真ん中
	right	右
	length	長さ
タスク 28	coin	硬貨
	remain	残額
	index	添え字
	pattern	パターン，様式
	total	合計
タスク 30	score	成績
	result	結果
	point	点数
	rival	相手
	hit	命中
タスク 31	result	結果
	all	全体
	part	一部
タスク 33	input	入力
	binary	2 進数
	decimal	10 進数
タスク 34	target	標的
	left	左
	right	右
	pivot	中心点
	current	現在の
	temp	一時的な
	index	添え字
タスク 36	result	結果
	min	最小値
	max	最大値
	mode	最頻値
	average	平均値
	median	中央値
	length　25	長さ
	count	数を数える
	output	出力

表 14 「意味なし」変数の変数長分布

文字数	度数
1	4
2	0
3	2
4	9
5	11
6	12
7	2
8	1
9	3
10	0
11	3
12	2
合計	49

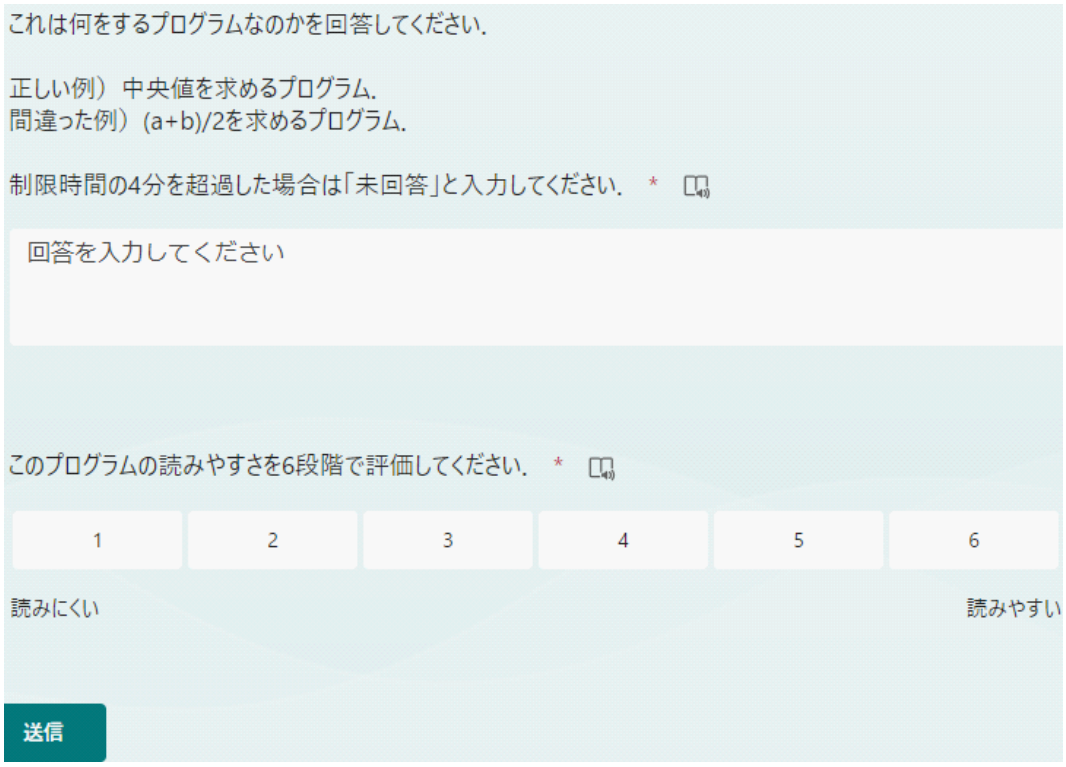

図 5 forms によるアンケート