一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

ソフトウェアメトリクスを用いた 高可読性コーディング能力の定量的評価

中村 優太† 上野 秀剛‡

†奈良工業高等専門学校電子情報工学専攻 〒639-1080 大和郡山市矢田町 22 番地
‡奈良工業高等専門学校情報工学科 〒639-1080 大和郡山市矢田町 22 番地
E-mail: †nakamura@info.nara-k.ac.jp, ‡uwano@info.nara-k.ac.jp

あらまし ソースコードを読む作業の時間的コストは大きく，ソフトウェア開発現場では可読性の高いソースコードを実装することが求められる．しかし教育現場では，学習者が書いたソースコードが可読性の観点から評価されることは少なく，可読性を考慮した実装能力を身につける事が難しい．本研究では，ソースコード全体の可読性を定量的に測定するソフトウェアメトリクスを提案する．本稿ではインデントとアルゴリズムの違いがソースコードの可読性に及ぼす影響を定量的に評価する．インデントによる可読性の評価では，ソースコードのインデント状態を場合分けし，被験者による可読性評価をもとに定量化する．アルゴリズムによる可読性の評価では，アルゴリズム全体の複雑さを表す拡張サイクロマティック数を提案する．被験者実験の結果，インデント状態の異なるソースコードに対する可読性の違いを表す重みを求めることができた．また，拡張サイクロマティック数と主観的な可読性の評価に相関がみられた．

キーワード ソースコード可読性，コーディング能力，コンジョイント分析，サイクロマティック数

1. はじめに

ソフトウェアライフサイクルにおいて，保守作業量は 70%と高い割合を占めると言われる[1]．さらに，保守作業の全行程の中で最も時間的コストの高い作業は，ソースコードを読み理解することであるといわれている[2]．可読性の高いソースコードを作成することはソフトウェア開発の生産性を高め，品質の向上にも寄与すると考えられる．

しかし，大学等のプログラミング教育の現場においては，プログラムが正しく動作するか評価されるものの，そのソースコードの可読性は評価されないことが多い．原因として，可読性は定量的な評価基準を設定しにくい要素であるからと考える．そこで本研究では，ソースコードの可読性を定量的に計測するソフトウェアメトリクスを提案し，定量的評価を可能にする．

可読性に影響を与える要素として変数やメソッドの名前や改行，コメント，インデントの有無，使用するアルゴリズムなどが考えられる．本研究では，インデントとアルゴリズムによる可読性を評価する．インデントは視覚的に可読性を高める効果があると考えられるが，プログラミング教育現場には初学者が多く，インデントを整えてコーディングしない場合が多いと考えられる．また，アルゴリズムが異なるとソースコード中に現れる制御フローも異なるため，読みやすさに影響を与えると考えられる．しかし，教育現場でのテストや課題では，可読性の評価が困難なことから動作の正誤のみを重視した実装を学習者が行っていると考えられる．学習者にメトリクスを用いてフィードバックすることで，可読性を意識したコーディングを習得できると考えている．

2. 関連研究

2.1. プログラミング能力の評価

プログラミング能力の評価に関する研究としては，プログラマのデバッグ能力に着目したものがある．松本らは，各エラーが作成されてから除去されるまでの時間的効率であるエラー寿命に着目している[3]．この研究では，能力の高いプログラマほどエラーを作り込まず，エラーが含まれたとしても短い期間でそれを取り除くことができるという考えに基づき，プログラミング能力を定式化している．高田らは，プログラマのデバッグ能力をキーストロークから測定する方法を提案している[4]．プログラマのデバッグ中の状態を「コンパイル」，「プログラム実行」，「プログラム変更」の 3 種類に分類した上で，デバッグ中はこの 3 状態の繰り返しであるとし，これらの 3 状態をキーストロークから検出する．デバッグ能力の高いプログラマは 3 状態の少ない繰り返しで，バグを取り除くという特徴からデバッグ能力を定量化している．

これらの研究はいずれもデバッグ能力を定量的に評価している．本研究でもプログラミング能力の 1 つとしてソースコードの可読性に着目し，高い可読性で実装する能力を定量的に評価する．

2.2. ソースコードの可読性評価

Buse らはソースコードの可読性を評価するために可読性メトリクスを提案している[5]．ソースコードの可読性は変数の数やコメント行数などの様々な要素に重み付けをした線形和で表せると仮定した．学生 120

人による Java ソースコードの主観的な読みやすさの評価から機械学習による定量化を行った.

Buse らの研究では,可読性に関わる多くの要素を機械学習によって 1 つのメトリクスで評価している.本研究は教育現場において学生に対するフィードバックを行うことを目的としている.そのため各要素に対する可読性を定量的に評価することで,学生がそれぞれの要素を意識してコーディングするよう支援する.

3. インデントによる可読性の評価

インデントはソースコード中のネスト構造を視覚的に表すことで可読性に影響を与える.本章ではインデントが可読性に与える影響を定量的に評価する.

3.1. 提案メトリクス

ソースコードのインデントを,ネスト構造(3 種類)とインデントの状態(3 種類)から 9 状態に分類し,各状態がソースコードの可読性に与える重みを求める.求めた重みを基に,プログラマが書いたソースコードのインデントによる可読性を定量的に表す.

3.1.1. ネスト構造

インデントはネスト構造の範囲を視覚的に表す.どの命令がどのネストに存在するのか一目でわかり,可読性を向上させる.一方で,適切に設定されていないインデントは,ネストの状態に対して誤解を生じてしまう可能性があり,可読性を低下させる.統合開発環境(IDE)の多くに自動でインデントを設定する機能があることや,コーディング規約が整備されていることで,開発現場においては適切なインデントが指定されることが多いと考えられる.一方で,教育現場においては,初学者が文法エラーを残したままプログラミングを進めるために IDE による自動インデントが機能しない場合や,コピー&ペーストを多用することで正しいインデントが設定されない場合がある.

本研究ではオブジェクト指向プログラミング言語における主要なネスト構造であるクラス,メソッド,ブロック(if 文や for 文)の 3 種類を対象に評価する.

3.1.2. インデント状態

インデントの状態を「あり」,「なし」,「混合」の 3 状態に分類する.図 1 にあるブロックにおける 3 状態の概要を示す.「あり」はソースコードのあるネスト構造において,すべての場所でインデントがされている状態,「なし」はソースコードのあるネスト構造において,すべての場所でインデントがされていない状態,「混合」とはインデントがされている場所とされていない場所が混在している状態を表す.インデント状態

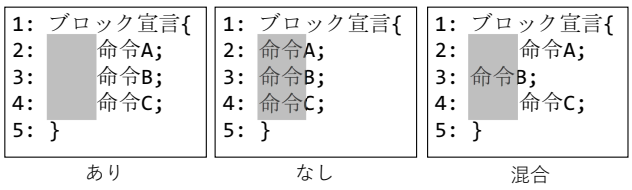

図 1:インデントの 3 状態

「あり」によってネスト構造の範囲を資格的に表すことができる.しかし,「なし」ではネスト構造の範囲を視覚的に理解することはできず,「あり」よりも可読性が下がると考えられる.「混合」では,インデントがされている行とされていない行が混じっており,統一感の無さからソースコードを読む者に混乱を与え,さらに可読性が下がると考えられる.

3.1.3. 重みの算出

ソースコード中に現れるインデントをネスト構造と状態の組み合わせ 9 種類とみなし,各状態が可読性に与える重みを求める.重みの算出にはコンジョイント分析[6]を用いる.ある事物や事象に対して,被験者がどのような要素を重視するか分析する手法で,マーケティング等の分野で広く用いられる.コンジョイント分析は比較したい複数の要素を組み合わせて作成した 1 つの状態を表すコンジョイントカードを複数用意する.ある基準を元に被験者が各カードにつけた順位を基に,1)どの状態がどの程度重視されているか(部分効用値),2)どのネスト構造がどの程度重視されているか(重要度),3)各コンジョイントカードに示された状態が表すソースコードのインデントによる可読性(全体効用値)を算出する.

本研究の重み(部分効用値)の算出では,ネスト構造とインデント状態を組み合わせて作成したコンジョイントカードに対して被験者の主観により順位付けを行う.順位データをコンジョイント分析することで各ネスト構造のインデント状態が可読性に与える重みを定量的に求めることができる.また全体効用値を求めることで,コンジョイントカードが示すソースコードの可読性を定量的に評価できる.

3.2. 実験

インデント 9 種類に対する重み付けを求めるための被験者実験を行う.ネスト構造 3 種類とインデント 3 状態に対して,$L_9(3^3)$ 型直交表に基づいて作成したソースコード 9 種類(表 1)をコンジョイントカードとする.ソースコードは同じ仕様,同じアルゴリズム,同じ構文で作成された,インデントのみが異なる Java のプログラムである.プログラムの仕様は「5 人分の国語と数学と英語の成績を入力し,最後に各教科の平均点を表示する」ものである.

表 1：コンジョイントカード

	クラス	メソッド	ブロック
カード 1	あり	あり	あり
カード 2	なし	なし	あり
カード 3	混合	混合	あり
カード 4	なし	あり	なし
カード 5	混合	なし	なし
カード 6	あり	混合	なし
カード 7	混合	あり	混合
カード 8	あり	なし	混合
カード 9	なし	混合	混合

被験者は本校情報工学科に在籍の 18 歳から 22 歳の計 17 名で，いずれも Java を使用した，複数のクラスを作成するプログラミングの授業を受講済みである．実験は複数の被験者を対象に同時に行う．被験者には日本語で記述された仕様と，9 種類のソースコードを印刷した計 10 枚の紙を配布し，ソースコードを可読性が高いと感じる順番に順位をつけるよう指示する．制限時間の 16 分間が経過した後，被験者に結果集計用紙を配布し，結果を記入してもらう．得られた順位データに対してコンジョイント分析を行い，部分効用値，重要度，全体効用値を求める．またアンケートで被験者のプログラミングレベルを 3 段階に分類する．

3.3. 結果と考察

表 2 に部分効用値を示す．各ネスト構造においてインデント状態「あり」が最も部分効用値が高く，「混合」が最も低い．これは，ソースコードを読む上で「あり」が最も可読性が高く，「混合」状態が最も低いことを表す．

ここで，プログラミング能力が高い人は経験的に効率的な読む方法を身につけている一方で，能力が低い人は読み方が身についていないと考える．そのため，プログラミングレベルによって重視する特徴が異なる可能性がある．学習者のソースコードを上級者の観点に基づいて評価する事で，上級者と同様の観点を身に着けられると考えられる．そこで，アンケート結果から被験者を上級者，中級者，下級者の 3 段階に分けてレベル別の重要度を分析する．分類基準を以下に示す．

- 下級者： 499 行以下のソースコード実装経験あり（授業で必ず達成するレベル）
- 中級者： 999 行以下のソースコード実装経験あり（授業成績優秀者が達成するレベル）
- 上級者： 1000 行以上のソースコード実装経験あり（授業外の活動が必要なレベル）

表 2：9 状態の部分効用値

ネスト構造	状態	部分効用値
クラス	あり	0.96
	なし	0.31
	混合	-1.27
メソッド	あり	1.51
	なし	0.00
	混合	-1.51
ブロック	あり	1.39
	なし	0.18
	混合	-1.57

レベル別の重要度を図 2 に示す．下級者はクラス，メソッド，ブロックの重要度がほぼ同じだった．中級者はブロックのインデントの重要度が最も高く，メソッド，クラスの順で重要度が高い．上級者は中級者とは反対にクラスのインデントの重要度が最も高く，メソッド，ブロックの順で重要度が高い．

下級者はブロックインデントを重視する被験者とクラスインデントを重視する被験者がおり，結果として重要度がほぼ均等になったと考えられる．中級者はソースコードの細かな流れを重視するため，ブロックを読む際のインデントを重視すると考えられる．上級者は先にクラス内のメソッドの位置など全体の構成を先に読むことが言われている[7]．上級者はトップダウンに大まかな流れを重視して読んで行くと考えられる．そのため，クラスのインデントを最も重視し，メソッド，ブロックの順に重視していると考えられる．

下級者，中級者，上級者では重視するネスト構造が違うことがわかる．そこで，上級者が感じる可読性を定量的に評価するために，上級者のデータから 9 状態の部分効用値を求めた結果を表 3 に示す．各ネスト構造において「あり」が最も高く，「混合」が最も低い．

求めた部分効用値から全体効用値を求めることにより，各状態の組み合わせによるソースコードの可読性を定量化する．式 1 を用いて全体効用値を求める．

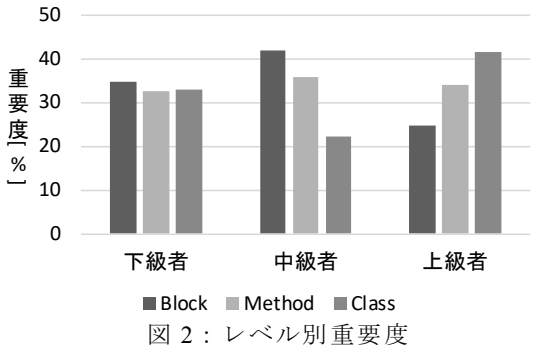

図 2：レベル別重要度

表 3：9 状態の部分効用値（上級者）

ネスト構造	状態	部分効用値
クラス	あり	1.67
	なし	0.50
	混合	-2.17
メソッド	あり	1.58
	なし	0.00
	混合	-1.58
ブロック	あり	0.83
	なし	0.58
	混合	-1.42

全体効用値
= クラスインデント(あり*or*なし*or*混合)
+メソッドインデント(あり*or*なし*or*混合)
+ブロックインデント(あり*or*なし*or*混合)
(1)

例として，評価対象であるソースコードのインデント状態がクラス「あり」，メソッド「混合」，ブロック「あり」であった場合，それぞれの部分効用値を合計した 0.92 が可読性となる．また，評価対象であるソースコードのクラス，メソッド，ブロックのインデント状態全てが「なし」である場合，可読性は 1.08 となり，「あり」と「混合」が混在したソースコードよりも可読性が高いと評価できる．教育現場においてソースコードを評価する場合には，全体効用値の最大値（すべて「あり」: 4.08）を 100 点，最小値（すべて「混合」: -5.17）を 0 点として正規化することで学習者へのフィードバックが容易になる．

4. アルゴリズムによる可読性の評価

ある仕様を満たすプログラムでもアルゴリズムが異なると制御フローが異なる場合がある．あるアルゴリズムがもつ制御フローが他のアルゴリズムよりも複雑であった場合，理解が難しくなるため，ソースコードの可読性を下げると考えられる．本章ではアルゴリズムによる可読性を定量的に評価する．

4.1. 論理構造と可読性

メソッドの複雑さを表すメトリクスである McCabe のサイクロマティック数を拡張し，アルゴリズム全体の複雑さを表す数値として用いる．サイクロマティック数（以下，cyc と表す）は 1 メソッドの制御パス数を表す複雑度メトリクス[8]であり，制御フローのリンク数とノード数から表される（式 2）．

$$cyc = リンク数 - ノード数 + 2 \quad (2)$$

例として，図 3 にあるメソッド A のソースコードとその制御フローを示す．制御フローにおける命令の書かれた 1 つの箱をノード，ノード間をつなぐ矢印をリンクとすると，例のメソッドの cyc は 3 となる．直感的にはサイクロマティック数は「制御フローにおける閉じたループの数 +1」で表すことができる．分岐やループが多いほどメソッドのサイクロマティック数が大きくなる．

しかしサイクロマティック数はメソッド単体の複雑さを評価するものであって，複数のメソッドからなるソースコード全体の複雑さを評価するものではない．すべてのメソッドのサイクロマティック数を足し合わせる(sum-cyc)方法やソースコード内で最大のサイクロマティック数(max-cyc)を複雑さとする方法が考えられる．しかし sum-cyc は同じアルゴリズムで実装されているソースコードに対してもメソッド分割の数によって異なる数値で評価する．またアルゴリズムが複雑であってもメソッド数を増やし，すべてのメソッドの cyc を小さくなるようにメソッド分割すれば max-cyc は小さくなる．sum-cyc や max-cyc は同じアルゴリ

```
 1: private void methodA(){
 2:   int sum = 0;
 3:   int x = 0;
 4:   int a[] = {18, 29, 36, 12};
 5:   for(int i = 0; i < 4; i++){
 6:     int x = a[i];
 7:     if(x > 0){
 8:       sum = x + sum;
 9:       methodB();
10:     }
11:   }
12: }
```

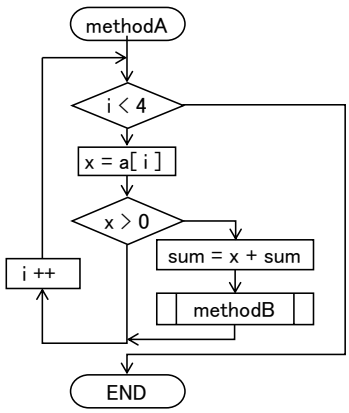

図 3：ソースコードと制御フロー

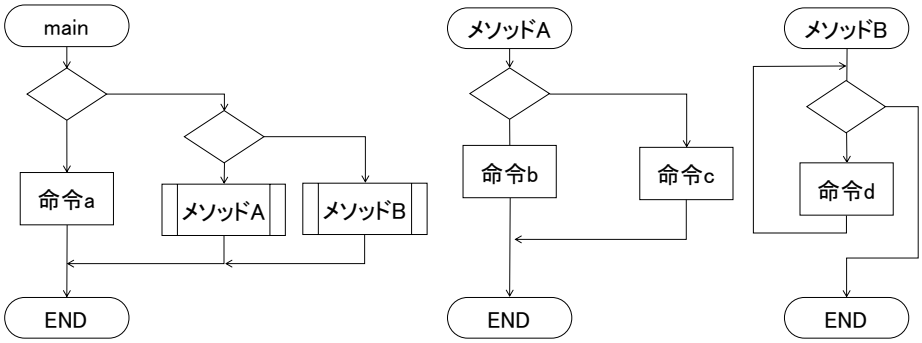

図 4：3 つのメソッドからなるプログラムの制御フロー

ズムで実装されているにも関わらず，メソッド分割方法によって異なる複雑さを表し，アルゴリズムによる複雑さを表す指標として適していない．アルゴリズムの複雑さを正確に表すためには，メソッドへの分割方法に依存しない計測方法が必要である．

そこで本研究ではサイクロマティック数を拡張し，ソースコード全体の複雑さを表すメトリクスとして利用する．拡張サイクロマティック数(ex-cyc)は複数メソッドからなる一連の処理がもつ制御フローのサイクロマティック数である．例として，図 4 のような制御フローを持つプログラムを考える．このプログラムはメインメソッドから 2 つのメソッドが呼び出されている．この 3 つのメソッドからなるアルゴリズムの制御フローを統合し，1 つの制御フローで表すと図 5 のようになる．統合した制御フローに対し計算したサイクロマティック数を ex-cyc と定義する．図 5 の場合，ex-cyc は 5 となる．ex-cyc はメソッドの分割方法にかかわらず，アルゴリズムに依存した複雑さを表す．

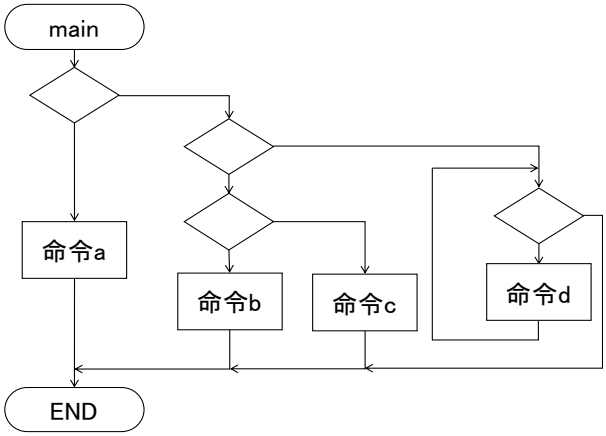

図 5：統合した制御フロー

4.2. 実験

拡張サイクロマティック数と可読性の相関を被験者実験により評価する．被験者は本校情報工学科に在籍の 19 歳から 22 歳の計 10 名で，いずれも Java を使用したプログラミングの授業を受講済みである．実験は複数の被験者を対象に同時に行う．

被験者には同じ仕様を異なるアルゴリズム（異なる拡張サイクロマティック数）で実装した 4 つソースコードを読んでもらう．プログラムの仕様は「2 つの数字列を入力し，2 つ目に入力された数字列が 1 つ目に入力された数字列を昇順に並び替えたものであれば”OK”，それ以外なら，”WRONG ANSWER”と表示する」ものである．被験者には実験開始時に仕様を提示し理解してもらう．プログラムは Java で実装されている．ソースコードは紙に印刷し，4 つのソースコードを 1 つずつ順番に被験者に提示する．被験者は 1 つのソースコードを 10 分の間に読み，可読性を 0 点から 100 点の範囲で採点する．4 つのソースコードへの採点終了後，被験者に結果集計用紙を配布し，結果を記入してもらう．またアンケートにより，被験者のプログラミングレベルを 3 段階に分類する．

被験者の採点基準を統一するため，事前に 2 つのソースコードを読んでもらい，1 つを 100 点，もう 1 つを 0 点の基準とする．基準となる 2 つのソースコードにも 10 分間の読む時間を設ける．基準となる 2 つを含んだ，6 つのソースコードの ex-cyc と行数とメソッド数を表 4 に示す．100 点の基準となるソースコードは ex-cyc が最小のも大のものとした．評価対象となる 4 つのソースコードの ex-cyc は 6 から 11 である．ソースコードは不自然なメソッド分割や意味のない命令（使わない変数宣言や意味のない if 文など）を入れることなく，異なるアルゴリズムを用いて実装することで ex-cyc を変化させている．

ex-cyc が大きいほど，アルゴリズムは複雑であり，可読性は低くなると考えられる．そのため，ex-cyc と被験者の評価値には負の相関があると考えられる．

4.3. 結果と考察

ex-cyc と被験者が採点した可読性の評価値との相関係数は-0.55 とやや強い負の相関が認められた．実験では同じ仕様をもつプログラムを対象としているため，仕様の理解しやすさとは別に，アルゴリズムが複雑な

表 4：実験で使用するソースコードの ex-cyc

ソースコード	ex-cyc	行数	メソッド数
100 点基準	6	45	1
0 点基準	23	92	1
No.1	6	55	5
No.2	7	64	5
No.3	9	67	3
No.4	11	60	1

プログラムほど，可読性が下がることを表す．

アンケートにより被験者のプログラミングレベルを下級者，中級者，上級者に分けてレベルごとに分析した．レベルの基準は 3.3 節に示したものと同様にした．レベルごとの ex-cyc と可読性の評価値との相関係数を表 5 に示す．プログラミングのレベルが高いほど強い負の相関が見られ，上級者のほうが下級者より ex-cyc が高いほど読みにくいと感じていることを表している．上級者は複雑なアルゴリズムのソースコードを読んでいる際，同じ処理を実現できる，より単純な（ex-cyc が低い）アルゴリズムを思いつき，そのアルゴリズムと比較して複雑なアルゴリズムに対して低い評価をしていると考えられる．一方で下級者は評価対象の 4 つのソースコードに対して一律に高い評価をしており，相関係数が低くなった．本実験ではあらかじめ仕様を被験者に伝えており，ソースコードの理解は容易だったと考えられる．下級者は理解した上でアルゴリズムを読みにくさという観点から比較を行うまでに至らず，読めたか読めなかったかで評価し，可読性についてこのような結果になったと考える．

また，上級者は拡張サイクロマティック数と可読性の評価値に-0.85 と強い負の相関があることから上級者が感じる可読性を表すメトリクスとして拡張サイクロマティック数は有用であると考えられる．拡張サイクロマティック数によりプログラミング上級者の基準でソースコードのアルゴリズムによる可読性を定量的に評価でき，教育現場でフィードバックを行うことで学習者は開発現場で活きる高可読性コーディングを習得することができる．

5. おわりに

本研究ではソースコードの可読性を定量的に評価する方法を提案した．視覚的な可読性の評価として，ソースコードのインデントをネスト構造とインデントの状態をもとに 9 状態に分類し，各状態が可読性に与える影響を定量化した．被験者実験の結果，インデント状態の異なるソースコードに対する可読性の違いを表す重みを求めることができた．また，論理構造的な可読性の評価としてアルゴリズムの複雑さに基づいた可読性の定量化を行った．サイクロマティック数を拡張し，ソースコード全体の複雑度を表す拡張サイクロマティック数(ex-cyc)を定義した．被験者実験の結果，ex-cyc と可読性の間に負の相関が認められ，可読性を表すメトリクスとして有用であることを示した．

表 5：レベル別相関係数

プログラミングレベル	相関係数
下級者	-0.33
中級者	-0.48
上級者	-0.85

本研究の結果を用いることで，プログラマの作成したソースコードの可読性を定量的に評価できるようになり，教育現場などで可読性についての評価を定量的に学生にフィードバックすることが可能となる．定量的なフィードバックを受けた学習者は可読性をより意識してコーディングするようになると期待される．

本研究の今後の課題として，変数名やコメントなど可読性に影響を与えると考えられる他の要素についての可読性を定量化することがあげられる．あらゆる要素の可読性を評価が可能になると，それらの評価を組み合わせて可読性の総合評価が可能となる．また，本研究では被験者として学生を対象としたが，より開発現場に近いプログラマで行うことで上級者が感じる評価で定量化でき，その評価を学生にフィードバックすることで学生は開発現場で活きる高可読性コーディング能力を習得することができると考えている．

文　献

[1] Barry B., Victor R. B., “Software Defect Reduction Top 10 List,” Computer, Vol.34, No.1, pp.135-137 (2001).

[2] Andrew J. K., Myers B. A., Coblenz, M. J., and Aung, H. H., “An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks,” IEEE Trans. Softw. Eng., Vol.32, No.12, pp.971-987 (2006).

[3] 松本 健一，井上 克郎，菊野 亨，鳥居 宏次，“エラー寿命に基づくプログラマ性能の実験的評価－大学環境におけるプログラム開発－,”電子情報通信学会論文誌 D, Vol.J71-D, No.10, pp.1959-1965 (1988).

[4] 高田 義広，鳥居 宏次，“プログラマのデバッグ能力をキーストロークから測定する方法,”電子情報通信学会論文誌 D-I, Vol.J77-D-I, No.9, pp.646-655 (1994).

[5] R. Buse and W. Weimer, “Learning a Metric for Code Readability,” IEEE Trans. Softw. Eng.,Vol.36, No.4, pp.546-558 (2010).

[6] 菅 民郎, “Excel で学ぶ多変量解析入門,”オーム社 (2004).

[7] 鷹治 沙織，上野 秀剛，“コードレビュー時の読み方教示によるレビュー効率の変化,”電子情報通信学会技術研究報告． KBSE, 知能ソフトウェア工学 Vol.114, No.128, pp.1-8 (2014).

[8] T. McCabe, “A complexity measure,” IEEE Trans. Softw. Eng., Vol.2, No.4, pp.308–320 (1976).