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

信学技報
TECHNICAL REPORT OF IEICE.

プログラミング演習における初学者を対象とした コーディング傾向の分析

伏田　享平† 　玉田　春昭†† 　井垣　　宏††† 藤原　賢二† 　吉田　則裕†

† 奈良先端科学技術大学院大学 情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
†† 京都産業大学 コンピュータ理工学部　〒 603–8555 京都府京都市北区上賀茂本山
††† 大阪大学大学院 情報科学研究科　〒 568–0871 大阪府吹田市山田丘 1–5
E-mail: †{kyohei-f,kenji-f,yoshida}@is.naist.jp, ††tamada@cse.kyoto-su.ac.jp, †††igaki@ist.osaka-u.ac.jp

あらまし　高等教育機関等で実施される初学者を対象としたプログラミング教育において，学習者が陥りやすい誤りを事前に把握することは重要である．一方で，これまでプログラミング初学者がどのようにソースコードを編集し，誤りに陥っているか，その傾向は十分に明らかになっていない．本稿ではプログラミング演習時に取得した初学者のソースコード編集履歴を分析し，その傾向を明らかにする．分析にあたってはプログラミング熟練者の目視による定性的分析と，ソースコードの編集に関わるメトリクス計測に基づく定量的分析を実施した．分析の結果，初学者が陥りやすい誤りのパターンを観測した．また，演習において理解困難な状態に陥っている場合，ソースコードの編集距離に着目したメトリクスの変動にある種の傾向があることを確認した．

キーワード　プログラミング教育，メトリクス，リポジトリマイニング

Coding Pattern Analysis for Novice Programmer in Programming Exercise

Kyohei FUSHIDA†, Haruaki TAMADA††, Hiroshi IGAKI†††,

Kenji FUJIWARA†, and Norihiro YOSHIDA†

† Graduate School of Information Science, Nara Institute of Science and Technology
Takayama-cho 8916–5, Ikoma-shi, Nara, 630–0192 Japan
†† Faculty of Computer Science and Engineering, Kyoto Sangyo University
Motoyama, Kamigamo, Kita-ku, Kyoto-shi, Kyoto, 603–8555 Japan
††† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita-shi, Osaka, 568–0871 Japan
E-mail: †{kyohei-f,kenji-f,yoshida}@is.naist.jp, ††tamada@cse.kyoto-su.ac.jp, †††igaki@ist.osaka-u.ac.jp

Abstract　It is important for instructors to comprehend common pitfalls for novice students in programming education at institute of higher education. Giving well-timed advice to students who fall into a pitfall is able to realize efficient learning and keep up their motivation. However, so far, it is less well-understood how novice students edit source code and then fall into a pitfall. In this study, we analyze the processes of editing source code by novice students in programming exercise, and then identify the characteristics of analyzed processes. This analysis comprised of qualitative and quantitative parts based on observation by experienced programmers and measurement of metrics related with editing source code, respectively. As a result, we confirmed the patterns of common pitfalls for novice students, and the characteristics of transitions of edit distance metric in the case that novice students faced difficulty in understanding the exercise.

Key words　Programming education, metrics, repository mining

1. はじめに

ソフトウェアはあらゆる製品に組み込まれており，その重要性は増す一方である．その中で優秀なソフトウェア開発技術者の養成は，高品質なソフトウェア開発を進める中で重要な要素の一つである．ソフトウェア開発技術の中でも，その実装技術であるプログラミングに関する教育は多くの教育機関で実施されている．

プログラミング教育を行う際，プログラミングを初めて学ぶ学習者（以下，初学者と呼ぶ）はプログラミングの構文や概念などの理解不足によりしばしば誤る．誤りを引き起こす原因としては，与えられた課題内容の理解不足や，学習対象のプログラミング言語の理解不足が考えられる．このような誤りに陥っている際，指導者は誤りを早急に把握し，対処する必要がある．適切な対処が行われなかった場合には，初学者は演習内容を理解できず学習意欲を失う可能性もある．一方で，プログラミング演習では数名の教員が多数の受講生を対象に指導をすることが多い．そのため，全受講生の状況を逐次把握し学習状況に応じた対応を行うことは困難である．

そこで，初心者が陥りやすい誤りについて，どのような誤りが発生しているか，その傾向を分析・分類することは，適切な教育を行っていくうえで有益である．事前にこのような傾向を把握しておくことで，初学者が誤りを起こした際に指導者は適切なタイミングで対応を行うことができる．また，初学者に対し事前にこのような傾向を提示しておくことで，初学者自身で未然に誤りを回避することが期待できる．

本稿では初学者のコーディング過程を分析し，その傾向を明らかにすることを目的とする．分析にあたっては，初学者を対象としたプログラミング演習実施時に,「ファイル保存時」「ソースコードのコンパイル時」といった細かい単位でソースコードのスナップショットを取得した．このスナップショットを対象に定性的および定量的な分析を行った．定性的な分析としては，コンパイル時のエラーメッセージとその際のソースコードをもとに，初学者が課題の要求に沿った編集を行っていたかを目視で分析した．定量的な分析としては，ソースコードのトークンに基づく編集距離に着目し，その値の傾向を分析した．

2. 分析の進め方

本稿では,「プログラミング演習において，学習者がどのように課題に取り組んでいるか」をコーディング傾向と呼ぶ．これまでプログラミング演習において，初学者がどのように課題に取り組んでいるかについては十分明らかになっていない．本稿では初学者のコーディング傾向を明らかにするため，実際にプログラミング演習で収集されたデータを対象に，定性的および定量的な分析を行う．以下に各分析の方針について示す．

2.1 熟練者による定性的な分析

定性的な分析では，プログラミング初学者が具体的にどのようにして課題に取り組んでいるかを，十分な経験を持つプログラミング熟練者によって目視で分析する．定性的な分析の概念図を図 1 に示す．分析にあたって，コンパイルを実行した際のコンパイル対象のソースコードと，コンパイルの結果（エラーメッセージ）を用いる．この 2 つの入力をもとに,「前回コンパイル時よりどのような修正を行ったか（修正の意味的内容）」「行った修正は妥当であったか（修正の妥当性）」をプログラミング熟練者によって判断する．この判断結果を用いて，妥当な修正が行われなかったとき初学者はどのような誤りに陥っていたか，その修正の内容を確認する．

定性的分析を行う意図は，コンパイラの出力（エラーメッセージ）だけでは判断することの出来ない，プログラムの意味的な誤りも含めて初学者がどのような誤り方をするかを明らかにするためである．また，課題を達成できたかどうかだけではなく，課題に取り組んでいる間どのような誤り方をしたか，そしてどのようにして誤りに気づき修正していったかを明らかにしたい．そこで，上記のようにプログラミング熟練者が目視によってどのような編集を行ったか判断を下した．

2.2 定量的な分析

定量的な分析では，プログラミング初学者がコーディングを行う過程で作成したソースコードを対象に，プロダクトメトリクスを計測し，その傾向を分析する．計測にあたり，その対象となるソースコードは定性的な分析と同じくコンパイル時にその時点でのスナップショットを取得する．本稿では学習者のソースコード編集履歴をより細粒度に計測するため,「トークン編集距離」と呼ぶメトリクスを導入する．

図 2 にトークン編集距離算出の概念図を示す．トークン編集距離は 2 つのソースコードを入力とし，字句解析を行った後，2 ソースコード間のレーベンシュタイン距離を算出することで求まる．レーベンシュタイン距離は，2 つの文字列間の不一致度を表す値である．具体的には一方の文字列から他方の文字列に変形するのに要した文字の挿入，削除回数の最小値で求められる．

トークン編集距離を求めるにあたっては，ソースコードを字句解析しソースコードのトークン化を行う．本研究で用いたトークン化は下記のルールに従う．

- 空白文字，コメントは全て無視する．
- 識別子，予約語，リテラルは 1 文字に符号化する．その他はそのままとする．
- トークンの種類（型名やリテラル）やスコープの違いは

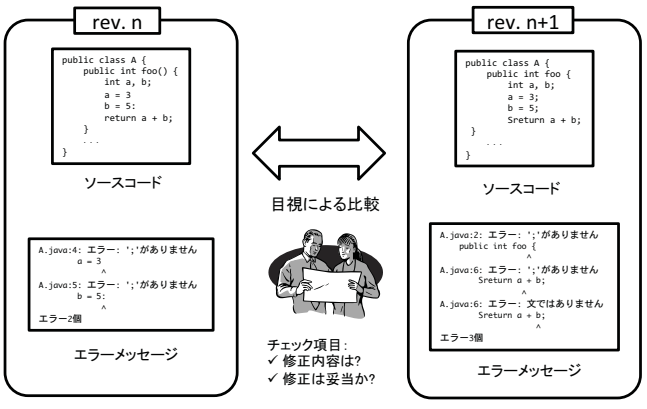

図 1　定性的な分析の概念図

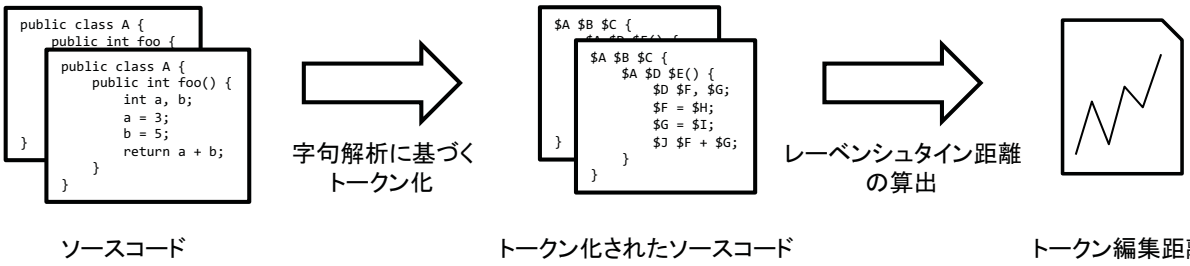

図 2　トークン編集距離の算出

考慮しない．

例えば図 3(a) に示すようなソースコードは，本稿で考えるトークン化を行うと図 3(b) のようになる．(ただし，読みやすさを考慮して図 3(b) において空白文字は無視していない．) このようにトークン化した 2 つのソースコード間において，レーベンシュタイン距離を計測することでトークンレベルでの編集距離を算出することが可能となる．

メトリクスを利用した分析の意図は，初学者が誤りに陥っていることをソースコードの意味的解釈を行うことなく検知したいためである．例えば初学者が課題の内容理解に手間取っている場合には，ソースコードに対する編集が一切行われていないことが考えられる．また，正解となるソースコードが与えられた際，前回取得したスナップショットと着目しているスナップショットの間でトークン編集距離が増加しているにも関わらず，正解となるソースコードとのトークン編集距離が変わらない場合，初学者は誤った修正を行っている可能性が考えられる．このようにソースコードとその編集履歴を対象に計測できる定量的な値を時系列に観測する．

3. コーディング傾向の分析

2. 節で述べた方針に従って定性的および定量的な分析を行った．今回の分析の目的は，以下の 2 点である．

- 初学者が陥りやすい誤りのパターンを明らかにする
- 初学者が誤る際の学習状況，およびソースコードの定量的な傾向を明らかにする．

以下，分析対象としたデータと分析手順について述べる．

3.1 分析対象

表 1 に分析対象としたデータセットの概要を示す．本稿では 2 つのデータセットを対象に分析を行った．データセット A，B ともに，対象となるプログラミング言語に対する事前知識を持

```
public class A {
    public int foo() {
        int a, b;
        a = 3;
        b = 5;
        return a + b;
    }
}
```

(a) トークン化前

```
$A $B $C {
    $A $D $E() {
        $D $F, $G;
        $F = $H;
        $G = $I;
        $J $F+$G;
    }
}
```

(b) トークン化後

図 3　本手法で用いるトークン化の例

表 1　分析対象データセット

データセット名	対象言語	学生数	課題数	スナップショット数
データセット A	C 言語	3	10	191
データセット B	Java	37	7	6428

たない初学者が対象となっている．初学者に与えられる課題は，それぞれの言語が備える基本的な機能で解答可能であり，特殊なライブラリなどを用いる必要は無い．また，課題にはあらかじめ雛形となるソースコードが示されている．初学者は雛形のソースコードに対して，与えられた仕様に基づき，欠落している機能を追加，修正することで解答する．解答にあたっては制限時間が設けられており，初学者は一定時間内にソースコードを作成する必要がある．

データセット A は京都産業大学の情報系学部 3 年生を対象に取得した．データ取得には第 2 著者らが開発したプログラミング教育支援フレームワーク [1] を利用している．このフレームワークは GCC のラッパとして動作しており，学習者がソースコードのコンパイルを行う際に，その時点でのソースコードのスナップショットとコンパイル結果（エラーメッセージ）を自動的に保存する．データセット A は通常のプログラミング演習の授業を行った際のデータではなく，上記フレームワークの評価実験時のデータを用いている．

データセット B は東京工科大学の情報系学部 1 年生を対象としたプログラミング演習の授業時に取得した．データ取得には第 3 著者らのグループが開発したコーディング過程可視化システム [2] を利用した．このシステムは Web システムとして実装されており，ソースコードを開く，保存する，コンパイルする，実行するといったイベントが発生した際に，自動的にその時点でのソースコードのスナップショットを保存する．また，1 分ごとにシステムが自動的にスナップショットを保存する．表 1 のスナップショット数には，このようにシステムが自動的に (ユーザが意図せずに) 保存したものも含まれる．今回はこのシステムが取得したデータのうち，授業 2 回分のデータのみを分析対象とした．このデータセットに関しては，模範解答や授業中にティーチングアシスタントが学習者にどのような支援を行ったかを記録した情報（指導記録）も入手した．

本稿では定性的分析にデータセット A を利用した．これは定性的分析を行うにあたり，目視で判断可能なデータ量であったためである．一方，定量的分析はデータセット B に対して実施した．データセット B は先に示した通り，定量的な傾向を検証

するのに必要な模範解答や指導記録が利用可能である．

3.2 分析手順

2. 節で示した方針に従って，下記に示す手順で分析を行った．

定性的な分析　データセット A について，ソースコードの修正内容とその妥当性について，人手により分析を行った．

定量的な分析　データセット B について，ソースコードの編集履歴を利用してソースコードメトリクスを計測した．また，各種統計量を合わせて算出した．

以降，各手順について詳細を示す．

3.2.1 定性的な分析

定性的な分析は，第 1 著者及び第 4 著者（以降，分析者と呼ぶ）が目視により実施した．分析者はどちらも C 言語を対象とした初学者向けプログラミング演習における指導経験を有している．分析に際しては，各ソースコードのスナップショットに対して下記の情報を目視で確認した．

- 着目しているスナップショット（ソースコード）
- GNU diff を利用した前回編集時との差分情報
- コンパイル結果（エラーメッセージ）
- プログラム実行結果（コンパイルに成功した時のみ）

上記 4 種類の情報より，下記に示す 3 つの観点から分析を行った．

不具合原因　ソースコードに不具合が含まれる場合，その内容を自然言語により記述した．

修正内容　diff の内容からどのような修正を行ったか，自然言語により記述した．

修正の妥当性　修正内容が課題の意図に沿ったものになっているか．もしくは，ソースコードの不具合を混入しているか．

3.2.2 定量的な分析

定量的な分析では，まず全スナップショットについて下記の 3 つの観点からトークン編集距離を求めた．(注1)

（1） 模範解答とのトークン編集距離

（2） 自身が提出した解答とのトークン編集距離（正解したもののみ）

（3） 前回取得したスナップショットとのトークン編集距離

次に，上記 3 つの観点で求めたメトリクスについて，各学習者，課題ごとに順位相関を求めた．最後に，先に求めた相関係数で特徴的な値を示していたものに着目し，演習時の指導記録などをもとに詳細な分析を行った．

4. 結果・考察

4.1 コーディング時の誤りパターン

表 2 に各学習者が取り組んだ課題数とその中で完成に至ったものの数，および編集回数を示す．また編集の中で，分析者が題意に沿って編集を行っている，もしくはプログラム中に欠陥を混入するような修正をしていると判断したものについて，その回数を示している．

今回データセット A の学習者は，C 言語の学習前に Java によるプログラミング演習の授業を履修していた．また，ユーザ A2 に関しては Ruby などのスクリプト言語を事前に自習していた．これらの点を踏まえ，以下に分析の結果見られた傾向について示す．

表 2　定性的な分析の結果

ユーザ名	課題数	完成課題数	編集回数		
			妥当な編集	欠陥混入	計
ユーザ A1	10	3	20	38	92
ユーザ A2	10	10	19	7	44
ユーザ A3	10	10	15	13	24

修正すべき箇所とは関係ない箇所を修正し続ける

特にユーザ A1 で多く見られた事象として，本来修正すべき箇所には手を加えず，課題内容とは関係の無い箇所を編集していることが確認された．プログラミング熟練者であれば，エラーメッセージなどをもとに修正箇所の絞り込みを行うことが考えられる．しかし，初学者の場合はエラーメッセージの情報を十分に理解できず，修正箇所を突き止めることが困難であるため試行錯誤的に修正を行っていると考えられる．

このような初学者に対しては，コンパイラが出力するエラーメッセージの意味と，具体的な修正箇所を絞り込むための戦略をあらかじめ教育しておくことが有用であると考えられる．また，4.2 節では初学者がこのような行動をしたと仮定して，定量的な分析結果をもとに初学者が誤りに陥ってることを検出するための方法について触れる．

自分の有しているプログラミング言語の知識に引きずられている

他の言語で利用できる機能や文法を C 言語でも利用しようとしてエラーを引き起こしている傾向が観測された．その中でも多く見られたのが文字列の結合である．Java であれば String 型の文字列は加算演算子により結合することが可能である．しかし，C 言語では加算演算子による文字列の結合はサポートされていない．

初学者はこれまでに自分が獲得したプログラミング言語の仕様と異なることを理解するまでに，プログラミング熟練者よりも時間がかかると思われる．既に他の言語を学習した経験のある者に対しては，習得済み言語との相違点を事前に提示することで，効率的に教育できる可能性がある．

4.2 トークン編集距離に着目した分析

図 4 にデータセット B を対象に計測した，3.2.2 で示した (1) から (3) のメトリクスの傾向を示す．図 4 は学習者および課題ごとに，システム上に記録されたソースコードのスナップショットに対する (1) から (3) のメトリクスと時間変化について，ケンドールの順序相関係数を求め，その分布をメトリクスごとに示したものである．以下では演習時の指導記録も併用し，各メトリクスについて特徴的な値を示したデータについて考察する．

メトリクス (1) に関しては，多くのデータが負の相関を示している一方で，強い正の相関を示しているものもある．ただ，このように強い正の相関を示しているものについて詳細な分析

(注1)：トークン編集距離の取得にあたっては，Repository of JavaCC grammars [3] で公開されている Java 1.5 用の文法ファイルを修正し，この文法ファイルをもとに JavaCC [4] を用いてパーサを作成し利用した．

を行ったところ，多くの場合正しく解答できていた．これは，与えられた課題の仕様は満たしており正しく解答しているが，模範解答のソースコードとは構造が異なるためと考えられる．このようなケースの例を図 5 に示す．出題時には背景が白色となっている箇所のみが学習者に提示され，背景が灰色となっている箇所を補充する形で解答するよう出題された．

メトリクス (2) では，初学者自身が提出したソースコードの中で正解であったものと，各時点におけるスナップショットを比較しているため，全体として強い負の相関関係にあることが確認されている．当然のことではあるが，時間経過に従いメトリクス (2) は 0 に漸近していく傾向にあるためである．一方で，メトリクス (2) の中でも強い正の相関を示しているものがある．このデータについて分析をしたところ，一度正解となるソースコードを提出した後，次の課題に取りかかる際，誤って前の課題のソースコードを修正していたためであった．

先の定性的な分析にて観測された「修正すべき箇所とは関係ない箇所を修正し続ける」事象に初学者が陥ったとき，前回スナップショット取得時からのトークン編集距離は一定の値を示すにも関わらず，正解となるソースコードからのトークン編集距離は減少しないことが予想される．図 6 に示すように，実際

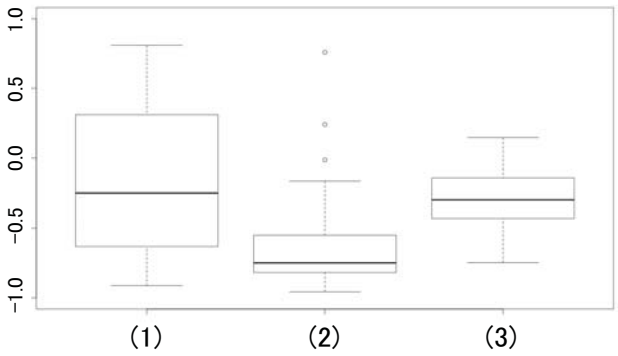

図 4　時間変化に伴う各メトリクスの相関係数の分布

```
class Report13_1a2
{
  public static void main(String[] args) {
    String str = "148737438777432";
    int x = Integer.parseInt(str.substring(0,4));
    x += Integer.parseInt(str.substring(5,9));
    if(x % 3 == 0)
      System.out.println("OK");
  }
}
```

(a) 模範解答

```
class Report13_1a2
{
  public static void main(String[] args){
    String str = "148737438777432";
    String str2 = str.substring(0,4);
    String str3 = str.substring(6,10);
    int y = Integer.parseInt(str2);
    int z = Integer.parseInt(str3);
    int x = y+z;
    if(x % 3 == 0)
      System.out.println("OK");
  }
}
```

(b) 学生が作成した正答例

図 5　模範解答とのトークン編集距離が大きくなった例

にデータセット B においてもこのような事例が観測された．図 6 はある学生 X が課題 Y に取り組んでいる際のトークン編集距離の遷移を表している．横軸は時刻，縦軸はトークン編集距離であり，図中の実線はメトリクス (3)，点線はメトリクス (1) の遷移を表している．この図からは，16:45 頃から 16:55 頃にかけて大幅にソースコードを編集しているにもかかわらず，正答したソースコードとの編集距離が増加していることが見て取れる．このような場合はソースコードに対して適切でない編集を加えている可能性が高い．実際に指導記録には，17:02 頃に学生 X が解答に必要な機能を理解していなかったため指導を行ったことが記録されていた．

トークン編集距離を用いて初学者を支援する場合には，初学者が作成したソースコードと模範解答とのトークン編集距離の傾向を観察し支援を行うことが考えられる．ソースコード行数の追加，削除と比較して，トークン編集距離はより細粒度に初学者の編集状況を捉えることが可能となる．一方で，本稿におけるトークン編集距離の定義では，ソースコード上のメソッド定義の順序や識別子名の違いなどに大きく影響をうける．そのため，先の例にもあったように，学習者が課題の要求を満たしたソースコードを作成していたとしても，模範解答とのトークン編集距離は大きく離れたままである可能性がある．この問題に対しては，今後トークン編集距離をメソッド単位で計測するなどの改良を行う予定である．

5. 関 連 研 究

プログラミング演習を対象とした教育支援に関する研究はこれまで盛んに取り組まれてきた．長らはプログラミング演習時に学習履歴を収集し，そこから学習者の達成度を判断したり，アドバイスを与えたりすることができるシステム “ proGrep” を開発した [5]．proGrep では，学習者があらかじめソースコードの文字列検索条件とそれに対応する指示（アドバイス）を「パターン」として登録する．学習者がソースコードを作成した際，上記のパターンに適合するものがあった場合，学習者にパターンを提示することで学習支援を行う．

藤原らは複数の時系列データを用い，プログラミング学習者の取り組み状況を定量的に把握する手法を提案してる [6]．この手法ではソースコード行数やコンパイル回数などをグラフ上に可視化する．また，グラフ上の特徴を定性表現としてモデル化し，これに従い抽出される特徴的な部分を定量化する．

田口らは個々の学習者の理解状況と学習意欲に応じて演習課題を出題する手法を提案している [7]．この手法では，協調フィルタリングを用いて過去の他の学習者の演習結果から学習者に最適な演習課題を選択，出題する．提案手法を実装したシステムを用いてプログラミング演習科目に適用した結果，学習者の学習継続率が向上したことが報告されている．

本稿でも利用しているレーベンシュタイン距離に関しては文字列の類似度を測るために用いられる．上田らはレーベンシュタイン距離に基づく類似度を用いて，プログラミング演習におけるレポート剽窃の検出を試みている [8]．

プログラミング時の開発者の行動に着目した研究もこれまで

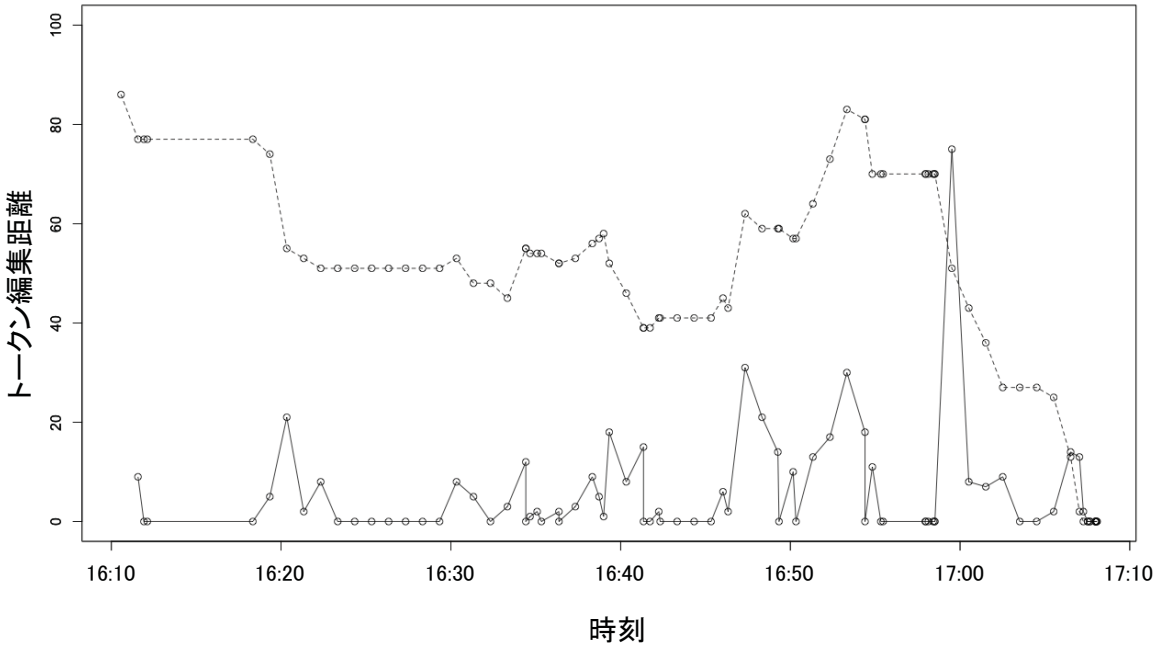

図 6　正答したソースコードからの編集距離と前回編集時からの編集距離の関係

行われている．Uwano らはソースコードレビューを行う際の開発者の視線の動きに着目した分析を行っている [9]．この研究では，欠陥の埋め込まれたソースコードをレビューする前に，ソースコード全体を概観した方が短時間で欠陥を検出する傾向にあることが報告されている．

吉村らはブレークポイントの使用履歴に着目してデバッグ作業の行動を分析している [10]．この分析では，デバッグ熟練者と初心者の行動の違いを分析した結果，熟練者はブレークポイントを用いた実行を頻繁に行う傾向にあることを報告している．

これらの研究に対して，本稿ではプログラミング演習で作成されたソースコードを事後分析し，プログラミング演習実施時に汎用的に適用できる可能性のあるパターンの抽出を試みている．また，定量的な尺度を導入し，初学者の学習状況をリアルタイムに把握するための手法の検討を行っている．

6. おわりに

本稿ではプログラミング初学者が誤りに陥る際の傾向を明らかにすることを目的として，プログラミング演習における初学者のコーディング過程を分析した．分析にあたっては，プログラミング熟練者による定性的な分析と，トークン編集距離を利用した定量的な分析を行った．定性的な分析では，初学者が誤りに陥っている際の行動パターンが観測された．定量的な分析では，トークン編集距離の変動を確認することで初学者が誤りに陥っているかを検出できる可能性があることを確認した．

今後の課題としては，今回検出されたパターンの他に初学者が陥りやすい誤りパターンが無いか，分析を進める必要がある．また今回導入したトークン編集距離など，定量的な分析に基づく初学者の学習支援が可能か検証を重ねていく予定である．

謝辞

本研究は日本学術振興会 科学研究費補助金 若手研究（B）（課題番号:20700029），研究活動スタート支援（課題番号:22800040, 22800043），京都産業大学総合学術研究所の助成を得た．

文　献

[1] H. Tamada, A. Ogino, and H. Ueda, “A framework for programming process measurement and compiling error interpretation for novice programmers,” Proc. 2011 Joint Conference of the 21stInternational Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement (IWSM/Mensura 2011), pp.233–238, Nov. 2011. Nara, Japan.

[2] 齊藤　俊，山田　誠，井垣　宏，楠本真二，井上亮文，星　徹，“プログラミング演習における受講生支援のためのコーディング過程可視化システムの提案，” 電子情報通信学会技術研究報告，March 2012.

[3] “Repository of JavaCC grammars”. http://java.net/projects/javacc/downloads/directory/contrib/grammars

[4] “JavaCC home”. http://javacc.java.net/

[5] 長　慎也，筧　捷彦，“proGrep-プログラミング学習履歴検索システム，” 情報処理学会研究報告，Vol.2005，no.15，pp.29–36，Feb. 2005.

[6] 藤原理也，田口　浩，島田幸廣，高田秀志，島川博光，“ストリームデータによる学習者のプログラミング状況把握，” 電子情報通信学会第 18 回データ工学ワークショップ，D9-5，March 2007.

[7] 田口　浩，糸賀裕弥，毛利公一，山本哲男，島川博光，“個々の学習者の理解状況と学習意欲に合わせたプログラミング教育支援，” 情報処理学会論文誌，vol.48，no.2，pp.958–968，Feb. 2007.

[8] 上田和志，富永浩之，“類似性に基づくレポート剽窃の検出ツールのプログラミング課題への適用，” 情報処理学会研究報告，Vol.2010，no.9，pp.1–8，Nov. 2010.

[9] H. Uwano, M. Nakamura, A. Monden, and K. ichiMatsumoto, “Exploiting eye movements for evaluating reviewer’s performance in software review,” IEICE Transactions on Fundamentals, vol.E90-A, no.10, pp.317–328, Oct. 2007.

[10] 吉村巧朗，亀井靖高，上野秀剛，門田暁人，松本健一，“ブレークポイント使用履歴に基づくデバッグ行動の分析，” 電子情報通信学会技術報告，Vol.109，no.307，pp.85–90，Nov. 2009.