Vol.2016-SE-191 No.16
2016/3/14

探索的プログラミング行動の自動検出によるモデル化の検討

槇原 絵里奈[1,a)]　井垣 宏[2]　吉田 則裕[3]　藤原 賢二[1]　飯田 元[1]

概要：探索的プログラミングとは，開発者が不慣れな言語や API を用いる場合などに，複数種類の実装を試行・評価しながら開発を進めていくサイクルを指す．我々は先行研究において，初学者が探索的プログラミングを行う上での問題点を明らかにし，支援のためのプログラミング環境 Pockets を開発した．本稿では探索的プログラミングモデルの構築を目的とする．まず初学者における探索的プログラミングの定義を明らかにし，次に Pockets を用いた探索的プログラミング行動の自動検出手法を提案する．

Investigating the Model of Automatically Detecting Exploratory Programming Behaviors

Erina Makihara[1,a)]　Hiroshi Igaki[2]　Norihiro Yoshida[3]　Kenji Fujiwara[1]　Hajimu Iida[1]

1. はじめに

探索的プログラミングとは，ソフトウェア開発において開発者が不慣れな言語や API を用いるときや，新しいアルゴリズムを用いる場合に，複数種類の実装をそれぞれ試行・評価しながら進めていくプログラミングのサイクルを指す [1]．このような試行・評価の繰り返しを続けていくことは，高品質なソフトウェア設計を行う上でも重要である [2]．また，開発者が探索的にプログラミングを進めることにより，対象に対する理解を深め，ソースコードの品質を改善することができる [3]．そのため，プログラミング初学者でも，新しい知識や技術について学ぶ際に探索的プログラミングを行うことが望まれる [2]．

プログラミング初学者がプログラミングを学ぶ場としてプログラミング演習があげられる．プログラミング演習とは，情報系学部学科を有する高等教育機関において開講されている，学生が実際にプログラミングを行う形式の授業を指す [4]．通常，プログラミング演習では，特定のプログラミング言語を扱うために必要な，機能のまとまりごとに段階的に教育を進めていく．プログラミング演習中は，教員からその日に取り扱うプログラミングの機能の扱い方に関する課題が学生へ与えられる．本研究では初学者が課題を解くためにソースコードへ対して行った編集，保存，コンパイル，実行などの，振る舞いをプログラミング行動 [5] と呼び，特に初学者が探索的プログラミングを用いて課題を解いている際のプログラミング行動を探索的プログラミング行動と定義する．

本研究では，探索的プログラミングを用いた初学者のプログラミング学習支援を目標とし，先行研究において探索的プログラミングをユーザへ促すためのプログラミング支援環境 Pockets の提案・開発を行った．本稿ではまず，これまでに行った実態調査から得られた結果を基に，初学者が行う探索的プログラミングの具体的な定義を明らかにする．そして，定義に従い Pockets を用いて探索的プログラミングを自動検出する手法を提案する．探索的プログラミングを自動検出することによって，エラーが続いていた箇所など，過去の探索的プログラミング行動をユーザに提示できる．ユーザである初学者は，いつどのようなソースコードの時にエラーが続いたか，または解消したのかを確認しながら，探索的プログラミングを行うことが出来る．さらに教育者は，初学者がプログラムを書く上で，どの箇所でつまづき，またどのようなアプローチで課題に取り組

[1] 奈良先端科学技術大学院大学 情報科学研究科
Nara Institute of Science and Technology
[2] 大阪工業大学 情報科学部 情報システム学科
Osaka Institute of Technology
[3] 名古屋大学 大学院情報科学研究科
Nagoya University
[a)] makihara.erina.lx0@is.naist.jp

もうとしているのかが分かる．以上より，初学者の探索的プログラミング行動の傾向が分かり，パターン抽出やモデル化が可能となると考える．

本稿の構成を以下に示す．2 章では先行研究によって得られた初学者が行う探索的プログラミングの特徴，およびそれを基にした初学者が行う探索的プログラミングの定義について述べる．3 章では Pockets を用いた探索的プログラミングの自動検出手法について述べる．4 章では 4 つのテストケースを基に，自動検出でどのような情報が得られるかについて述べる．5 章では自動検出の結果について考察を述べる．6 章をまとめとする．

2. 準備

本節では，本研究の支援対象である初学者が行う探索的プログラミングについて説明する．2.1 節では，初学者が行う探索的プログラミングの特徴について述べる．2.2 節では，我々が先行研究において開発した探索的プログラミング支援環境 Pockest について説明する．2.3 節では，初学者の探索的プログラミングを支援する上での課題について述べる．

2.1 初学者が行う探索的プログラミング

我々は先行研究において，学部一年生を対象とした Java のプログラミング演習で，学生がどのように探索的プログラミングを行っているか実態調査を行った [6]．一般的な探索的プログラミングは特定のアルゴリズムや API を対象とするが，この分析は初学者向けプログラミング演習が対象であったため，一般的な探索的プログラミングの定義よりも細かい粒度で探索が行われると予想した．よって，この分析では初学者が探索的プログラミングを行っているかの判断基準として「特定の行に対して，編集及びコンパイル，実行の一連のプログラミング行動が連続して複数回行われていること」と定めた．

実態調査により，我々は手戻り (Backtracking) を伴うものと伴わないものの 2 種類の探索的プログラミングを観測した．ここで手戻りとは，探索的プログラミングにおいて頻繁に観測される，開発者が自身のコードを以前と同一の状態に戻す振る舞いのことを指す [7]．手戻りは探索的プログラミングにおける特徴的な振る舞いである [8]．熟練の開発者の場合，まずプロトタイプと呼ばれる，要求の一部を実装したプログラムを作成する．このプログラムはコンパイル・実行時エラーを起こさず，変更の基点となる [3]．そしてプロトタイプへ新たな機能を実装し，コンパイルや実行を行う．この時にエラーが生じた場合，追加した機能にエラーが含まれている可能性が高いため，プロトタイプのソースコードへ手戻りを行い，別のアプローチで実装を続けることが出来る．

一方初学者の場合，エラーを含むソースコードも手戻りの基点となっていることが確認した．また，手戻りを行う際に必要以上に変数や文字を削除してしまったり，削除すべき箇所を削除せずに，新たなエラーが出現しているケースも見受けられた．手戻り先も数回前のリビジョンに戻る場合が多く，これは過去のコンパイルや実行に成功したソースコードを覚えきれていないためであると考えられる．

手戻りを伴わない探索的プログラミングは条件分岐や繰り返し処理などの条件式で多く行われていた．特に，$i == 0$ のような条件式の比較演算子部分で，頻繁に探索的プログラミングが使用されていた．これは，条件式に何かしらのエラーの原因があることが分かっているが，プログラムの挙動について理解していないため，演算子で考えられるものを順に代入，その都度コンパイルあるいは実行を行っているためであると考えられる．

2.2 探索的プログラミング支援環境：Pockets

2.1 節の結果を基に，我々は井垣らが開発した C3PV という web アプリケーション [9] を基に拡張する形で，探索的プログラミング支援環境 Pockets を実装した [10]．Pockets は図 1 に示すとおり 3 つの領域で構成されている．それぞれの領域について詳述する．

C3PV オリジナル領域 ユーザはこの領域でソースコードの記述が可能である．また，記述したソースコードは領域上部の Save，Compile，Run ボタンを押すことで，それぞれ保存，コンパイル，実行が可能である．コンパイルか実行ボタンを押した時，領域下部にその結果が出力される．

リビジョン一覧領域 C3PV オリジナル領域で Save, Compile，Run のいずれかのボタンが押された時，この領域にそれぞれの動作に対応した色のサムネイルを自動で表示する（リビジョン一覧表示機能）．枠線の色は Save，Compile，Run のいずれが実行されたのかを示しており，それぞれ黄色，青色，赤色に対応している．

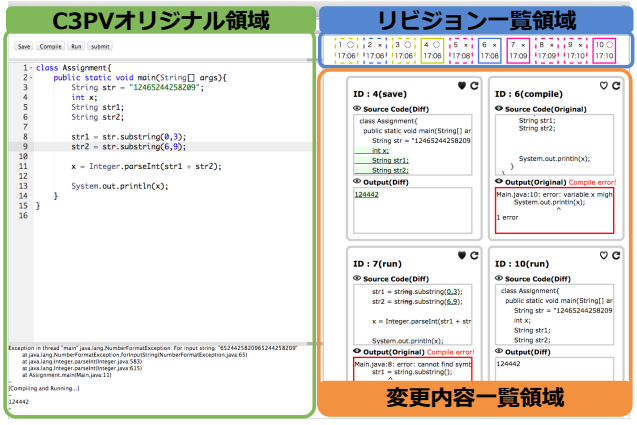

図 1 Pockest の UI

サムネイルには (1)ID，(2) コンパイル・実行結果の有無，(3) サムネイルが表示された時間の情報が含まれている．

変更内容一覧領域　リビジョン一覧領域でサムネイルが出現するのと同じタイミングで，この領域に過去と直近の，ソースコードと結果の差分を表示する差分表示領域が現れる（差分表示機能）．ユーザはソースコードと結果の差分を見ることで，自身がソースコードに対して加えた変更が結果にどのような影響を及ぼしたのかを確認できる．

差分表示領域の右上には矢印のボタンがあり，押すことで対応するリビジョンのソースコードが C3PV オリジナル領域のエディタ部分へ読み込まれる（手戻り機能）．手戻り機能はリビジョン一覧領域のサムネイルをクリックすることでも行うことが出来る．

ユーザは Pockets を用いて以下の手順で実装を進めていく．

手順 1　ユーザは C3PV オリジナル領域でソースコードのコーディングを行う．

手順 2　ユーザはコーディングが完了したら，目的に応じて Save，Compile，Run ボタンを押す．その時，リビジョン一覧領域にサムネイルが，変更内容一覧領域に差分表示領域がそれぞれ出現する．

手順 3　ユーザはサムネイルと差分表示領域を閲覧し，過去のリビジョンへ戻る必要があれば手戻り機能を使い，必要がなければ結果に応じてコーディングを続ける．

2.3　探索的プログラミングを支援する上での課題

Sandberg[3] は開発者が探索的にプログラミングを進めることにより，対象への理解を深めソースコードの品質を改善することができると述べている．さらに Myers ら [2] は，プログラミング初学者が新しい知識や技術について学ぶ際に，探索的プログラミングを行うことが望まれると述べている．つまり探索的プログラミングは，初学者がプログラミングを学習する初期段階から使用できるようになることが望ましい．

しかし，望ましい探索的プログラミングのサイクルについては，既存研究で言及されていない．探索的プログラミングはソースコードの頻繁な書き換えを行うため，1 つのプログラムを完成させるまでに時間がかかると予想される．Pockets には過去のソースコードへ手戻りする時間や，コンパイル・実行の回数を減らすために手戻り機能や差分表示機能を備えている．しかしながら，望ましい探索をユーザへ提案する機能は備えていないため，手戻り機能や差分表示機能を用いても探索に時間がかかる恐れがある．また探索の方法がわからない場合，ユーザは他者からのアドバイスを待つしかない．

望ましい探索的プログラミングのサイクルを確立するためには，より詳細な探索的プログラミングの分析が必要である．そこで，我々は初学者が行う探索的プログラミングの定義を定める．そして，Pockets を用いて探索的プログラミングを自動検出することで，探索的プログラミングのモデル化を目指す．

3.　提案手法

本章では，探索的プログラミングの定義および自動検出のためのアルゴリズム，そして Pockets を用いた探索的プログラミング検出結果の利用シナリオについて述べる．3.1 節ではこれまでの分析の結果に基づいて，初学者向け探索的プログラミングの定義を行う．そして 3.2 節では，前節で述べた探索的プログラミングの定義に基づき，探索的プログラミングを自動検出するためのアルゴリズムについて述べる．3.3 節では，自動検出によって得られるデータと，そのデータを Pockets を用いてどのように活用できるかについて述べる．

3.1　初学者向け探索的プログラミング定義

2.1 節で述べた分析の結果，初学者は，特定の演算子のみや if 文の条件式全体，あるいは実現したい機能を構成する複数行のまとまり，といった様々な粒度で探索的プログラミングを行っていることが判明した．つまり，行単位の変更のみに着目して探索的プログラミングが行われているか分析すると，全ての探索的プログラミングは検出できないと考えた．そこで，我々は初学者が行う探索的プログラミングについて「同一のブロック，行，あるいは行を構成する要素に対して修正及びコンパイル・実行結果の確認が連続で行われていること」と定義する．本稿で述べる修正とは，文や文字の挿入，削除，入れ替えを指し，ブロックとは通常中括弧 {} で囲まれた部分を表すが，ここではそ

```
 1:  public class Sample{
 2:   public int foobar(){
 3:    while(){
 4:     int x;
 5:    }
 6:   int y;
 7:    for(){
 8:     if(){
 9:      int z;
10:     }
11:    }
12:   }
13:   void fizzbuzz(){
14:   }
15:}
```

図 2　複数のブロックの深度を含むソースコード

のブロックを特徴づける前後の記述も含めるものとする．ブロックに含まれる記述例を以下に示す．

- ブロックの直前に記述されている条件式
- メソッドの宣言における戻り値とシグネチャの対

また，この定義ではこれまでの行単位での探索的プログラミングだけでなく，行内の演算子やリテラル，識別子，変数等の要素に対する連続的な変更や，if 文の条件式とブロック内記述等の細かい粒度から粗い粒度における探索的プログラミングが含まれる．

また，ブロックが入れ子になった場合を考慮するため，ブロックの深度を定義する．ブロックの深度とは，ブロックが入れ子になった場合，外側から数えて何番目にあるブロックであるかを表す．ブロックの深度の例を図 2 のソースコードを例えに示す．この場合，各ブロックの深度は以下のようになる．

深度 0　Sample（1 行目）
深度 1　foobar（2 行目）, fizzbuzz（13 行目）
深度 2　while（3 行目）, for（7 行目）
深度 3　if（8 行目）

例えば 6 行目の変数 x が複数回編集された場合，深度 2 の while 文の探索として扱う．すなわち，編集された箇所を内包する最小のブロックにおける探索として扱うものとする．

3.2　アルゴリズム

自動検出のアルゴリズムの概要図を図 3 に，手順について以下に示す．なお，本稿では隣接する編集履歴において，ソースコード間の変更はすべて 1 箇所であると想定している．

前提　編集履歴にあたるソースコードのスナップショットが 3 つ以上存在する．
手順 1　ソースコードの編集履歴中の，連続する 2 つのソースコードを比較する（図 3 では Rev.n-1 と Rev.n を示す）．
手順 2　2 つのソースコード間の変更箇所を特定する．
手順 3　手順 2 のソースコードをトークン化する．
手順 4　変更箇所の粒度を調べる．変更の粒度は以下のように分類される．
　粒度小　演算子や変数，識別子，リテラル等の文の構成要素 1 つ
　粒度中　1 つの文
　粒度大　複数の文
　最も細かい粒度のものが優先される．
手順 5　手順 2 において特定されたソースコードの変更箇所がどのブロックに含まれているか，全て特定する．
手順 6　最も深い深度のブロック名が選択される．
手順 7　手順 1 から 6 の作業を，次に連続するソースコード間でも行う（図 3 では Rev.n と Rev.n+1 を示す）．
手順 8　手順 6 において特定されたブロックが前の 2 点間のソースコードのブロックと一致するか確認する．

3.3　Pockets を用いた自動検出結果の利用シナリオ

Pockets はユーザが Save，Compile，Run ボタンのいずれかを押した際にソースコードがデータベースへ保存され，リビジョン一覧表示機能と差分表示機能が動作する．この時，探索的プログラミング行動を検出し，ユーザである初学者と教員に提示することが可能である．提示する内容は，(1) どのブロックが修正されたか，(2) その修正は前回の修正に対し探索的か否か，の 2 つがあげられる．これにより初学者は探索的プログラミングにおける手戻りを，構文情報も加味した上で，自身が戻りたいソースコードへ戻ることができ，教員は初学者がどのブロックに躓いているのかを知ることが出来る．

Pockets では初学者の探索的プログラミングの支援として，リビジョン一覧表示機能によるユーザの過去のプログラミング行動をサムネイルとして可視化している．サムネイルに含まれる情報の 1 つに，コンパイル・実行結果のエラーの有無がある．これは，コンパイル・実行を行った時，エラーが生じなかったら ○ を，エラーが生じたら × を表示するものである．この情報により，例えばエラーが続く前のソースコードに戻したい時，○ が表示されているサムネイルをクリックすることで，手戻り機能により初学者でも容易に過去のソースコードへ戻ることが可能となる．

ここで，サムネイルの情報に加え，探索を行ったブロックの情報も初学者に提示することで，初学者は手戻り先のソースコードをより的確に選択できると考える．例えば，エラーが続く前のソースコードへ単純に戻るのではなく，今ユーザが探索しているブロックを挿入した直後へ戻ることも可能になる．また，特定のブロックの探索を始めた時に新たにエラーが生じていたら，そのエラーの原因は探索を始めたブロックに含まれていることが分かる．このように，探索的プログラミング行動をユーザである初学者に提示することは，初学者が手戻りの際に混乱することなく本人が望むリビジョンへ手戻りをすることを可能にする．

Pockets 上で行われたソースコードの編集履歴，Save，Compile，Run のボタンを押した時間，実行結果やエラーの有無など，プログラミング行動は全てサーバ上のデータベースへ保存される．このデータはリビジョン一覧表示機能や差分表示機能としてユーザへ使用されるだけではなく，教員も同様に閲覧が可能である．そこで，初学者が行う探索的プログラミングに関する情報を教員へ提示することで，教員はプログラミング演習において，より学生の進捗に応じたアドバイスを与えることが可能となると考える．例えば，条件分岐，繰り返し処理の順序でプログラミング演習が行われるとする．繰り返し処理を題材とした演習時に，条件分岐について多くの学生が探索的プログラミ

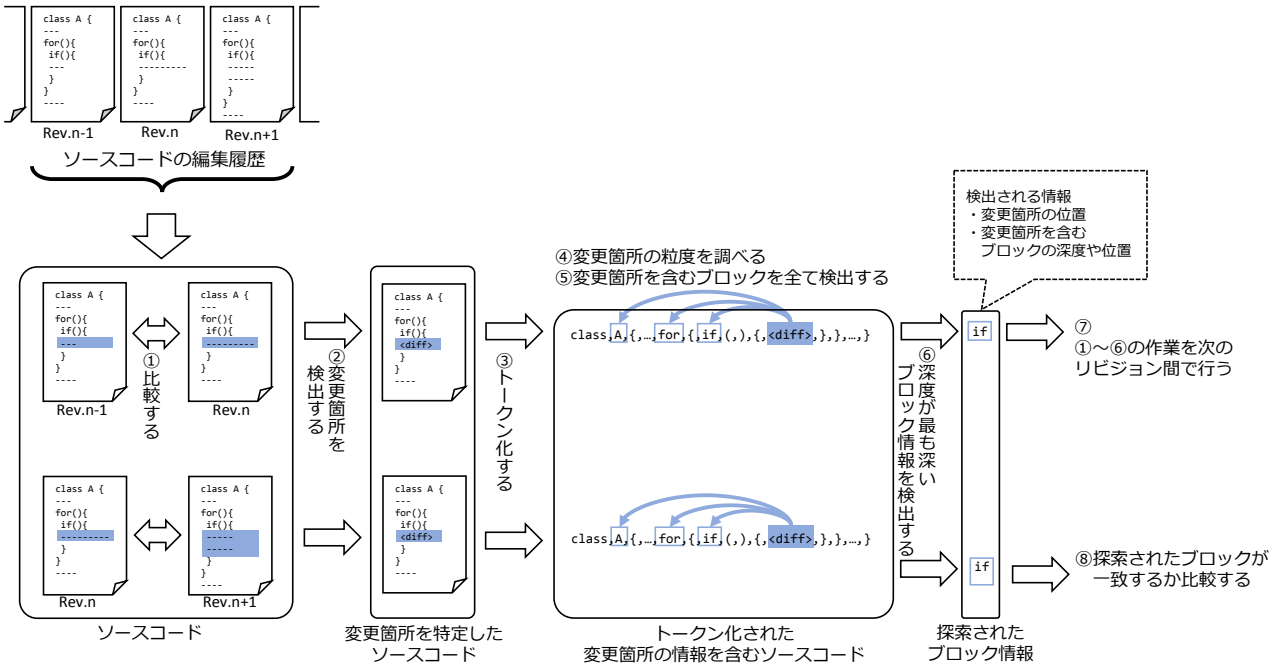

図 3 探索的プログラミング行動の自動検出アルゴリズム

ングを行っていた場合，これは前回の演習内容について十分に理解していない学生が多く存在することになる．したがって，教員は条件分岐について再び解説したり，追加で条件分岐に関する課題を与えたりする必要がある．従来，教員が学生の授業に対する理解度を測るには授業後の小テストやレポートを行うしかなかった．しかし，探索的プログラミングの対象となっているブロックを自動検出することで，教員は演習中に学生の躓いている要素が分かり，早期にアドバイスを与えることが可能となる．

4. ケーススタディ

本稿ではケーススタディとして，以下に示す 4 つのテストケースの探索的プログラミング行動の自動検出を Pockets を用いて行った．ケーススタディの目的は，正しいブロックが検出できるかを確かめるためである．

TC1 入れ子が連続で追加された時
TC2 複数のブロックが順に編集された時
TC3 同じブロック中の別の箇所が編集された時
TC4 同名のブロックが編集された時

4.1 各テストケースの概要

各テストケースについて以下に詳述する．各テストケースの例を示す図では，本ケーススタディに関係ない行や条件部分は省略している．全てのテストケースは (a) 探索前のリビジョンの時点で探索的プログラミングの自動検出が可能な状態であるが，(a) 探索前に書かれているブロック内はまだ探索されていないと仮定する．さらに，各テストケースの step 後には必ず 3.2 節で述べたアルゴリズムによる探索的プログラミングの自動検出が行われる．

TC1: 入れ子が連続で追加された時

入れ子が連続で追加された時の編集の例を図 4 に示す．この図では (a) 探索前から (b) 探索後のソースコードに至るまでに以下の編集が行われている．

step1 4 行目の for 文と対応する 8 行目の括弧を追加する
step2 5 行目の変数 x を追加する
step3 6 行目の if 文と対応する 7 行目の括弧を追加する

TC2: 複数のブロックが修正された時

複数のブロックが修正された時の編集の例を図 5 に示す．この図では (a) 探索前から (b) 探索後のソースコードに至るまでに以下の編集が行われている．

step1 5 行目の変数 foo を追加する
step2 7 行目の while 文と対応する 9 行目の括弧を追加する
step3 8 行目の変数 bar を追加する

```
   …
3:  while(){
4:
5:
6:
7:
8:
9:}
…
```

(a) 探索前

```
   …
3:  while(){
4:   for(){
5:    int x;
6:     if(){
7:     }
8:    }
9:  }
…
```

(b) 探索後

図 4 入れ子が連続で追加された時の例

TC3: 同じブロック中の別の箇所が編集された時

複数のブロックが修正された時の編集の例を図 6 に示す．この図では (a) 探索前から (b) 探索後のソースコードに至るまでに以下の編集が行われている．

step1 6 行目の System.out.println(y); を追加する
step2 4 行目の変数 y を追加する
step3 8 行目の System.out.println(y); を追加する

TC4: 同名のブロックが編集された時

同名のブロックが修正された時の編集の例を図 7 に示す．この図では (a) 探索前から (b) 探索後のソースコードに至るまでに以下の編集が行われている．

step1 5 行目の変数 fizz を追加する
step2 6 行目の変数 buzz を追加する
step3 8 行目の変数 fizzbuzz を追加する

```
   …
 3:  for(){
 4:   if(){
 5:
 6:   }
 7:
 8:
 9:
10:  }
…
```

```
   …
 3:  for(){
 4:   if(){
 5:    int foo;
 6:   }
 7:   while(){
 8:    double bar;
 9:   }
10:  }
…
```

(a) 探索前　　(b) 探索後

図 5 複数のブロックが編集された時の例

```
   …
 3:  for(){
 4:
 5:   if(){
 6:
 7:   }
 8:
 9:  }
…
```

```
   …
 3:  for(){
 4:  int y
 5:   if(){
 6:  System.out.print(y);
 7:   }
 8:  System.out.print(y);
 9:  }
…
```

(a) 探索前　　(b) 探索後

図 6 同じブロック中の別の箇所が編集された時の例

```
   …
 3:  if(){
 4:   if(){
 5:
 6:
 7:   }
 8:
 9:  }
…
```

```
   …
 3:  if(){
 4:   if(){
 5:   int fizz;
 6:   int buzz;
 7:   }
 8:  double fizzbuzz;
 9:  }
…
```

(a) 探索前　　(b) 探索後

図 7 同名のブロックが編集された時の例

表 1 探索的プログラミング自動検出の結果

TC	step	ブロックの深度	ブロック名	探索結果
TC1	step1	2	while	–
	step2	3	for	○
	step3	3	for	○
TC2	step1	3	if	–
	step2	2	for	×
	step3	3	while	×
TC3	step1	3	if	–
	step2	2	for	○
	step3	2	for	○
TC4	step1	3	if	–
	step2	3	if	○
	step3	2	if	×

4.2 探索的プログラミング自動検出の結果

テストケース 1 から 4 の検出結果を表 1 に示す．表には各テストケースの step1 から 3 によって行われた，変更が含まれるブロックの深度，ブロック名，行われた修正が探索的であったか否かの結果を ○ と × で表示している．それぞれ step1 では前回の変更がないため，探索か判定ができない．したがって各テストケースの step1 の探索結果には – を表示している．

検出結果を目視で確認したところ，全てのテストケースにおいて正しく検出できていることがわかった．また，本稿のケーススタディでは括弧の対応が取れたソースコードを対象に調査を行ったが，例えば括弧の開きと閉じが対応していないソースコードやコンパイルエラーや実行時エラーが生じるソースコードでも，今回対象にした探索的プログラミング定義に沿ったブロックを抽出できた．これは，エラーが多い初学者のソースコードにも対応できることを示している．

5. 考察

本稿ではまず初学者が行う探索的プログラミングの定義を定めた．そして，探索的プログラミング支援環境 Pockets を用いて，ユーザがソースコードを保存・コンパイル・実行のいずれかを行った時に，ユーザが加えた変更が探索的プログラミングか否かを自動検出する手法を提案した．以下，5.1 節で探索的プログラミングを自動検出することによる教員と学生のメリットについて考察を述べる．5.2 節で探索的プログラミングのモデル化に向けた考察を述べる．

5.1 探索的プログラミングの自動検出による教員と学生のメリット

提案手法により行単位だけでなく，ブロック単位での探索的プログラミング行動の特定が可能となった．プログラミング演習はブロック単位で単元が分かれていることが多いため，ブロック単位で探索が行われているかを検出することで，教員は初学者がこれまでに学んだブロックを理解

しているのか把握するのが可能になると思われる．その日の単元で取り扱うブロックが頻繁に探索されている場合，初学者は与えられた課題を理解し，取り組んでいることが分かる．一方で，それまでに学んだはずのブロックが頻繁に探索されている場合，初学者はこれまでの授業に対し十分に理解していないことが考えられるため，教員は復習問題を増やしたり，過去に教えたブロックについて再度解説を行う必要があることが分かる．

さらに，ブロックの自動検出は初学者が教員や TA へアドバイスを求める時にも有益である．通常のプログラミング演習では，初学者が課題や実装でわからない箇所が存在した場合，初学者は挙手を行い教員や TA がその初学者の元へアドバイスを与えに向かう．その時，初学者はこれまで自身がどのように課題に取り組んでいたのかを説明する必要があるが，いつどのような編集を行いエラーが生じたのかなどの情報を思い出しながら説明することは困難であると考えられる．しかし，Pockets のリビジョン一覧表示機能にあわせてブロックの情報も提示することで，どのブロックでエラーがいつ生じたのか，またエラーが生じる直前のソースコードはどのようになっていたのかなどを教員や TA へすぐに提示することが出来る．教員や TA も質問を行った初学者がその日の単元の理解不足で質問をしたのか，あるいはこれまでに授業を行った単元の理解不足で質問を行ったのか判断が可能となる．

5.2 探索的プログラミングのモデル化に向けた議論

Pockets のリビジョン一覧表示と手戻り機能に加えて，探索的プログラミングの情報を提示することで，初学者はより細かい粒度で自身が戻りたいソースコードへ手戻りが可能となる．教員は，探索されているブロックの情報から，プログラミング演習における学生全体の進捗の把握が容易に行えるようになると考える．

初学者の探索的プログラミング行動のモデル化を行うには，初学者の探索的プログラミングの傾向を知ることが必要である．傾向を知るためには，初学者が望む過去のリビジョンへ的確に戻る必要がある．本稿で提案した探索的プログラミングの自動検出アルゴリズムは，ケーススタディの実験結果，探索しているブロックを正しく検出することができた．この情報を用いることで，初学者は過去に実装したブロックとそのコンパイル・実行結果のエラーの有無を考慮した上で，手戻りや実装を続けることが出来る．

しかし，実際の初学者のプログラミング行動では条件式の比較演算子部分のような粒度の細かい探索が頻繁に行われる．さらに，本稿では変更箇所は 1 箇所のみの想定で検出を行っていたが，実際は複数箇所の同時変更も十分考えられる．例えば図 6 の場合，初学者は 5 行目の if 文の動作を知りたいため，変数 y を if 文の前後と中で宣言・出力を行っている．この場合，自動検出により検出されるブロックは 3 行目の for 文になるが，実際この初学者は if 文の動作を確認しようとしている．したがって，より正確に探索的プログラミングを検出するためには同一変数の探索の検出が必要である．今後はより粒度の細かい探索の自動検出を実装し，得られた結果を初学者や教員へ提示することで，より精度の高い探索的プログラミングを支援することが可能となり，探索的プログラミング行動のモデル化へも繋がると考えられる．

6. おわりに

本稿では探索的プログラミングを用いた初学者のプログラミング学習支援を目的とし，そのために探索的プログラミング行動モデルの構築を目指した．そのために初学者が行う探索的プログラミングの定義を明らかにし，自動検出のアルゴリズムを提案した．さらに，既存研究において開発した探索的プログラミング支援環境 Pockets へ提案するアルゴリズムを実装し，Pockets 上で探索的プログラミングの一部粒度の自動検出が可能となった．自動検出によって得られた結果を初学者と教員へ提示することで，初学者はより細かい粒度の探索的プログラミングが可能となり，教員は学生の進捗度合いに応じたアドバイスを与えることが可能であると考えられる．今後は，更に細かい粒度で探索的プログラミングの自動検出を行い，精度の高い探索的プログラミング行動モデルを構築したいと考える．

参考文献

[1] Sheil, B.: Environments for exploratory programming, *Datamation*, Vol. 29, No. 7, pp. 131–144 (1983).
[2] Carnegie Mellon University : Variations to Support Exploratory Programming, `http://www.exploratoryprogramming.org/`.
[3] Sandberg, D. W.: Smalltalk and exploratory programming, *ACM SIGPLAN Notices*, Vol. 23, pp. 85–92 (1988).
[4] 独立行政法人情報処理推進機構：IT 人材白書 2014，独立行政法人情報処理推進機構 (2014).
[5] Vihavainen, A., Luukkainen, M. and Kurhila, J.: Using Students' Programming Behavior to Predict Success in an Introductory Mathematics Course, *Proc. of EDM*, pp. 300–303 (2013).
[6] 槇原絵里奈，井垣　宏，藤原賢二，上村恭平，吉田則裕，飯田　元：初学者向けプログラミング演習における探索的プログラミングの実態調査と支援手法の提案，日本ソフトウェア科学会第 21 回ソフトウェア工学の基礎ワークショップ，pp. 123–128 (2014).
[7] Myers, B. A., Oney, S., Yoon, Y. and Brandt, J.: Creativity Support in Authoring and Backtracking, *Proceedings of Workshop on Evaluation Methods for Creativity Support Environments at CHI*, pp. 40–43 (2013).
[8] Yoon, Y. S. and Myers, B. A.: Supporting Selective Undo in a Code Editor, *Proceedings of the 37th International Conference on Software Engineering*, pp. 223–233 (2015).
[9] 井垣　宏，齊藤　俊，井上亮文，中村亮太，楠本真二：プログラミング演習における進捗状況把握のためのコー

ディング過程可視化システム C3PV の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 330–339 (2013).
[10] 槇原絵里奈，藤原賢二，井垣　宏，吉田則裕，飯田　元：初学者向けプログラミング演習のための探索的プログラミング支援環境 Pockets の提案，情報処理学会論文誌，Vol. 57, No. 1, pp. 236–247 (2016).