情報処理学会論文誌 Vol.57 No.1 236–247 (Jan. 2016)

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

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

受付日 2015年4月10日, 採録日 2015年10月2日

概要：ソフトウェア開発では，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ことが多い．このようなプログラミングスタイルを探索的プログラミングと呼び，初学者の学習モデルとしても適しているといわれている．実際に初学者向けのビジュアルプログラミング環境などは探索的に開発を進めることを想定して開発されているものも多い．一方で，Java や C といったプログラミング言語を対象とした初学者教育において，探索的プログラミングの支援を目的とした研究・開発はあまり行われていない．我々は学生に対し，ソースコードの変更履歴を可視化およびリスト化し，ボタン 1 つで過去の特定のリビジョンに手戻りが可能な探索的プログラミング支援環境 Pockets を提案する．2 種類のケーススタディにおいて学生に Pockets を使用して課題を解いてもらった結果，Pockets を使用した場合に，使用しなかった場合と比較して探索的プログラミングを行う回数が増加したことが確認された．また，実施後のアンケートにおいて，38 名の学生のうち 21 名が，Pockets 固有の機能によって課題が解きやすくなったと回答した．

キーワード：探索的プログラミング，プログラミング演習，プログラミング教育，コーディング履歴可視化

Pockets: An Exploratory Programming Support Environment for Introductory Programming Exercises

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

Received: April 10, 2015, Accepted: October 2, 2015

Abstract: When software developers deal with unfamiliar programming language and its APIs, they often try and evaluate multiple types of implementation. Such programming style is called as exploratory programming. Since exploratory programming is also suitable for novice programmers, existing visual programming environments for them assume that they develop their programs in exploratory. On the other hand, only a few research aim to support exploratory programming in education that teaches novices programming languages (e.g., Java, C). In this paper, we propose a supporting tool 'Pockets' for exploratory programming in programming exercise for novices. Pockets visualizes previous revisions of source code written by novices. It supports them in reverting to a past revision. Through the case studies using Pockets, we have confirmed that students with Pockets perform exploratory programming more than those without Pockets. Moreover, the questionnaire after the experiment has revealed that Pockets' original function facilitates 21 out of 38 students to solve assignments easier.

Keywords: exploratory programming, programming exercise, programming education, coding process visualization

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[2] 大阪工業大学
Osaka Institute of Technology, Hirakata, Osaka 573–0196, Japan
[3] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8601, Japan
[a)] makihara.erina.lx0@is.naist.jp

1. はじめに

近年の IT 市場拡大の背景を受け，実践的な能力を持つ技術者の育成が国内外問わず望まれている．そのため，情報系学部・学科を有する高等教育機関ではプログラミング演習と呼ばれる，学生が実際にプログラミングを行う形式の授業が開講されている．

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．機能のまとまりには条件分岐や配列，繰返しなどが含まれており，本研究ではこれらを構文要素と定義する．学生には，解説に含まれる例題を少し改変した簡単な課題から，これまでに学習してきた要素を組み合わせなくてはならないような複雑な課題が与えられる．これらの課題は最初の数問は教員による解説や解答例が提示されることもあるが，基本的には周りに相談せず独力で解くことが求められている．

しかし，実際のプログラミング演習では，プログラミング初学者はコンパイルエラーや実行時エラーなど，異常出力の解決に多くの時間を費やす [1]．よって，初学者に対しては構文要素に対する知識だけでなく，エラーが生じたときどうするか，求めている出力と違う出力が行われたときどうしたらいいかなど，プログラミングを行ううえでの振舞いについても教育を行う必要がある．本論文では，課題を解くためにソースコードへ対し行った編集，保存，コンパイル，実行など，振舞いのことをプログラミング行動 [2] と呼ぶ．

本研究では，開発者によるプログラミング行動の 1 つとして知られる探索的プログラミング（Exploratory Programming）[3] を初学者に対して促進する手法を提案する．探索的プログラミングとは，開発者が扱い慣れていない API やプログラミング言語，新しいアルゴリズムを使用する際に，実装が曖昧である箇所を編集，そのつどコンパイル，実行を行い，徐々にプログラムを完成形に近づけていく手法である．初学者も，プログラミング演習において多用な試行錯誤をともないプログラムを完成させていくため，一種の探索的プログラミングを行っていると考えられる．したがって，効率の良い探索的プログラミングを行うための支援をすることで，初学者はより円滑にプログラミング演習を進めることができると考えられる．

そこで，本研究では探索的プログラミングを効率良く行うためのプログラミング環境 Pockets を提案する．Pockets は，(1) リビジョン一覧表示機能，(2) 手戻り機能，(3) 差分表示機能の 3 つの機能を持つ．リビジョン一覧表示機能により，保存，コンパイル，実行といった過去のソースコードへの操作履歴，およびその結果をユーザへ提示し，手戻り機能により編集中のソースコードを過去の特定のソースコードに容易に戻すことが可能となる．また，差分表示機能により過去と直近のソースコードの差分が提示され，この機能によりユーザはソースコードの変更が出力にどのような結果を及ぼしたのかを確認することができる．これらの機能により，ユーザは課題を解くにあたって試行錯誤を効率良く行うことが可能となり，プログラミングに対する理解を深めながら，課題を解き進めることができると考えられる．

本論文の構成を以下に示す．2 章では準備として，探索的プログラミングの説明と，本論文で対象とするプログラミング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．3 章では 2 章で得られた知見を基に，提案手法について詳述する．4 章では 2 つのケーススタディの内容およびその結果を，5 章では 4 章の結果に対する考察について述べる．6 章で関連研究について述べ，7 章をまとめとする．

2. 準備

本章では，我々が着目する探索的プログラミングについて説明する．まず，2.1 節において一般的な探索的プログラミングの定義と概要について，既存研究とあわせて説明する．次に，2.2 節で本論文で対象とする初学者が行う探索的プログラミングについて，定義および特徴について説明する．最後に，2.3 節で初学者が行う探索的プログラミングの問題点について述べ，支援を行うにあたっての要件を述べる．

2.1 探索的プログラミング

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，複数種類の実装を試行・評価しながら開発を進めていくことが多い．Sheil [3] は，このよう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スタイルを探索的プログラミングと定義している．Rosson ら [4] の実験では，開発者らの主要な振舞いとして，プログラミングを行い，実行・テストし，プログラムが開発者の意図した状態であるかを評価し，意図にそぐわない場合はコーディング対象のすべてあるいは一部分を削除してやり直すといったものが観測されている．また，Brandt ら [5] は，開発者らは複数の可能性を試行する目的で頻繁に自身のコードを以前の状態に戻す，と実験結果に基づいて述べている．

Sandberg [6] によると，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とされている．そのため，仕様が曖昧な場合や新しい言語，アルゴリズムを用いて開発を行う場合において，探索的にプログラミングを進めることが望ましい．同様に Myers らの研究グループ [7] は，複数の変更パターンを反復的に評価することは高品質

なソフトウェア設計を行ううえでも重要であると述べている．さらに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と述べている．

2.2 初学者による探索的プログラミング

我々は先行研究において，学部 1 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を分析した [8]．この分析では，初学者向けのプログラミング授業を前提とするため，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を対象としている．細かい粒度とは，通常の探索的プログラミングが特定のアルゴリズムや API を対象をしていることに対し，Loop 文や if 文の条件・中身といった構文要素に着目することを指す．本論文では初学者が行う細かい粒度の探索的プログラミングを狭義の探索的プログラミングと呼び，狭義の探索的プログラミングを「特定の構文要素に対して，編集およびコンパイル，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」と定義する．

実際の分析では，41 名の学生が受講した 1 回のプログラミング授業において，狭義の探索的プログラミングに分類される一連のプログラミング行動が平均して 16 回観測された．また，狭義の探索的プログラミングとして，手戻り (Backtracking) をともなうものとともなわないものの 2 種類が観測された．なおここで手戻りとは，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舞いのことを指す [9]．以降では我々の先行研究において観測された 2 種類の探索的プログラミングについて詳述する．

2.2.1 手戻りをともなう探索的プログラミング

図 1 に手戻りをともなう探索的プログラミングの変更例を示す．変更は R1→R2→R1→R3→R1→R4 の順序で行われた[*1]．この学生の場合，R1 でコンパイルを行った後，R2 から R4 にかけて 5 行目の変数 x の代入文を重点的に編集している．また，いずれのリビジョンでもコンパイルあるいは実行が行われており，かつ R1，R2，R3 のときはコンパイルエラーあるいは警告が生じている．

このような変更を行う理由として，過去と現在のソースコードの内容および結果を見比べるためであると考えられる．たとえば図 1 の場合，R1 では x の宣言が重複しているため宣言の重複を指す出力エラーが生じていた．この学生は変数 x に問題があると考え変数を x から x1 に変更している．しかし，次は x が初期化されていないため，変数の初期化を行っていないという警告が新たに生じた．ここでこの学生は先ほど得たエラーと違うエラーが出現したことに気づき，先ほどのエラーは何だったのかを知るために

R1
```
[[ ソースコード ]] – – – – – – – – – – – – – – – – – – – – – –
4:  int x;
5:  int x = (int)(math.random()*8);
6:  System.out.println(x);
[[ 実行結果 ]] – – – – – – – – – – – – – – – – – – – – – – – –
x is already defined
```

R2
```
[[ ソースコード ]] – – – – – – – – – – – – – – – – – – – – – –
5:  int x1 = (int)(math.random()*8);
[[ 実行結果 ]] – – – – – – – – – – – – – – – – – – – – – – – –
x might not have been initialized
```

R3
```
[[ ソースコード ]] – – – – – – – – – – – – – – – – – – – – – –
5:  int x = (math.random()*8);
[[ 実行結果 ]] – – – – – – – – – – – – – – – – – – – – – – – –
Possible loss of precision
```

R4
```
[[ ソースコード ]] – – – – – – – – – – – – – – – – – – – – – –
5:  x = (int)(math.random()*8);
[[ 実行結果 ]] – – – – – – – – – – – – – – – – – – – – – – – –
1
```

図 1 手戻りをともなう探索的プログラミングの例
Fig. 1 Example of exploratory programming with backtracking.

再度 R1 の状態へ手戻りしコンパイルを行っている．R1 から R3 へ遷移した後も，double 型を int 型に代入していることを指摘する，R1 とは別の警告が生じている．その後，この学生は自身が変更すべき箇所は，変数名でもランダム関数の呼び出し前の int でもないことが分かり，x の宣言が重複していることに気づき，R4 で正しい変更を行った．

2.2.2 手戻りをともなわない探索的プログラミング

図 2 に手戻りをともなわない探索的プログラミングの例を示す．変更は S1→S2→S3→S4 の順で行われており，この学生は 12 行目にあたる while 文の条件を集中的に編集している．また，いずれのリビジョンでもコンパイルあるいは実行が行われており，かつ S1，S2，S3 のときはコンパイルエラーあるいは何も出力されないといった出力ミスが生じている．

図 2 の場合，while 文の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，文法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，そのつどコンパイルあるいは実行を行っていることが分かる．このような変更パターンは for 文，if 文の条件式でも同様に行われていることを確認した．

このように初学者が行う狭義の探索的プログラミングには手戻り，すなわち過去の同一状態にソースコードを戻す

[*1] 実際には各リビジョン間で 5 行目以外の変更も行われたが，本論文では省略している．

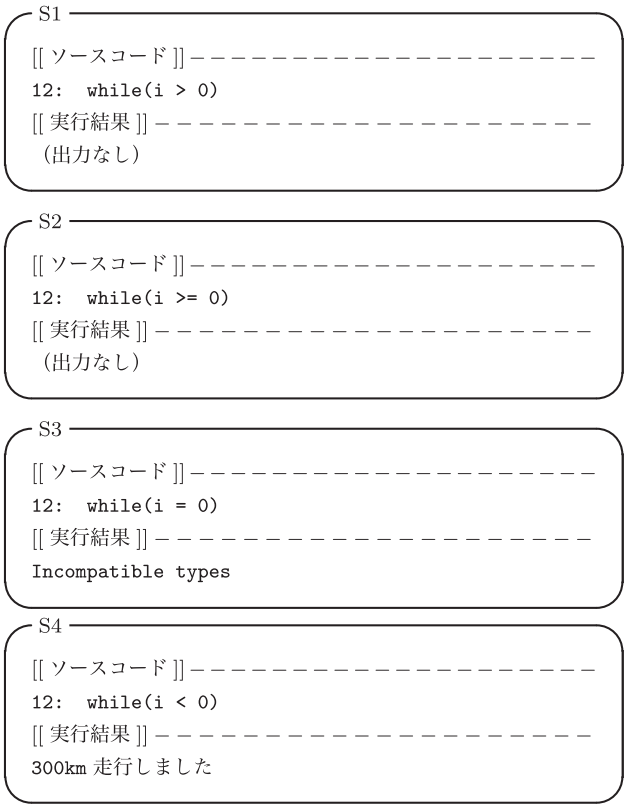

図 2 手戻りをともなわない探索的プログラミングの例
Fig. 2 Example of exploratory programming without backtracking.

ものと戻さないものがあることが我々の実験により観測された．

2.3 探索的プログラミングにおける課題

一般的な探索的プログラミングの課題として，探索の難しさがあげられる．手戻りをともなう，ともなわないにかかわらず，探索的にプログラミングを行うためには，1 度記述したソースコードを編集し，再実行する必要がある．そのため，修正時のバグ混入や意図した状態に手戻りができなくなるといった問題が発生することがある [10]．2.2 節で述べた先行研究では，180 分で 17 問の課題を解くことが求められるプログラミング演習において，16 名の学生が探索的プログラミングを実施しており，平均して 12.1 分程度費やしていることが確認できた．

これらの実態調査の結果や Sandberg [6] らの研究から，初学者の多くは探索的プログラミングを行っており，その際にソースコードが複雑になってしまい，混乱してしまうことがあるということが分かった．したがって，混乱を招くことなく探索的プログラミングを支援するツールがプログラミング初学者には必要であるといえる．

探索的プログラミングを行うにあたって，ソースコードのどの箇所を変更したか，またそれによって出力結果がどのように変化したかを一覧できることが望まれる．しかし，プログラミング演習ではテキストエディタとコマンドプロンプトを用いて演習を進める場合が多い [11]．この場合，過去のソースコードやその実行結果を知るためには，テキストエディタの Undo 機能を特定のソースコードまで繰り返し，再びコンパイルおよび実行をする必要がある．手戻りを行うときも同様である．Eclipse[*2] や Visual studio[*3] を使用している教育機関も存在するが，これらはプログラミング演習で使用する機能以外にも非常に多くの複雑な機能を持ち，初学者にとって望ましくないといえる [12]．

そこで，本研究では以下の要件を満たす探索的プログラミング支援環境を提案する．

R1 過去のソースコードの実行結果が一覧できる．
R2 特定の過去のソースコードへ容易に手戻りが行える．
R3 過去と現在のソースコードおよび結果の比較が行える．

特定のソースコードへ手戻りを行うためには，まず過去に自身がソースコードへ行った保存，コンパイル，実行といった動作，およびそれによって生じたエラーの有無が分かる必要がある（R1，R3）．さらにボタン 1 つで手戻りが行えることで（R2），たとえば求めている出力結果とは違うがエラーが生じないソースコードや，コーディングを始めたばかりの複雑化していないソースコードへ容易に戻すことができる．これらの機能により，探索的プログラミングを行ううえでのソースコードの冗長化，複雑化が緩和できると考えられる．また，手戻りの有無に関係なく，自身が加えたどの変更が結果にどのような影響を及ぼしたかを知ることができることが望ましい．

以上をふまえて，次章で我々の提案する探索的プログラミング支援環境について詳述する．

3. 探索的プログラミング支援環境：Pockets

我々は探索的プログラミング支援のため，以下の機能を持つプログラミング環境 Pockets を提案する．F1 から F3 のそれぞれの機能は 2.3 節の要件 R1 から R3 へそれぞれ対応している．

F1 リビジョン一覧表示機能
F2 手戻り機能
F3 差分表示機能

Pockets は，井垣らが開発した C3PV を基に拡張する形で実装している [13]．**図 3** に Pockets の UI を示す．Pockets は「C3PV オリジナル領域」「リビジョン一覧領域」「変更内容一覧領域」の 3 つの領域によって構成されている．なお，C3PV オリジナル領域はコントローラ領域，エディタ領域，出力領域の 3 つの小領域に分かれている．Pockets を用いたコーディングの流れを以下に示す．

手順 1 エディタ領域に表示されているオンラインエディ

[*2] https://eclipse.org/
[*3] http://www.microsoft.com/ja-jp/dev/default.aspx

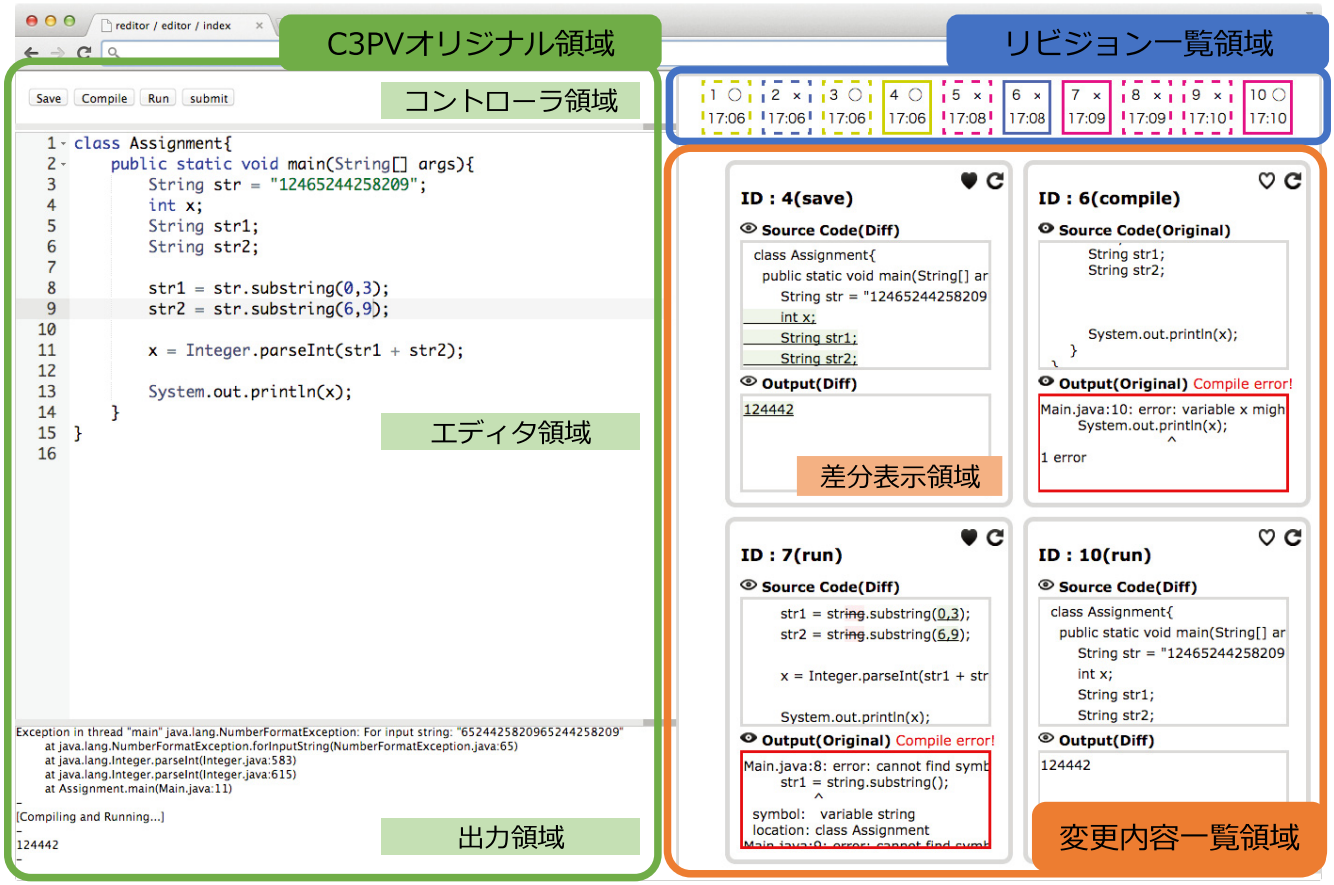

図 3 Pockets の UI
Fig. 3 A Pockets user interface.

タ[*4]を用いて課題のコーディングを行う．

手順 2 コントローラ領域にある保存，コンパイル，実行ボタンのいずれかを押す．それに対応した動作がソースコードに対して行われる．

手順 3 コンパイルあるいは実行結果が出力領域へ，サムネイル（3.1.1 項で説明）がリビジョン一覧領域へ，差分表示領域（3.1.2 項で説明）が変更内容一覧領域へそれぞれ出力される．ユーザは差分表示領域を見て，自身が加えた変更が結果にどのような影響を及ぼしたかを確認する．

手順 4 必要があれば手戻り機能を用いて過去のリビジョンへ手戻りを行う．

ユーザは手順 1 から 4 を繰り返すことで，課題のコーディングを進める．

3.1 システムの提供する機能

Pockets は C3PV のオリジナル領域に対し，新たに画面右上にリビジョン一覧領域が，画面右下に変更内容一覧領域がそれぞれ備わっている．リビジョン一覧領域ではリビジョン一覧表示機能を，変更内容一覧領域では差分表示機能と手戻り機能をそれぞれ提供する．

*4 http://ace.c9.io/

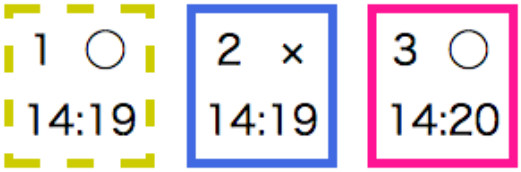

図 4 リビジョン一覧領域におけるサムネイルの例
Fig. 4 Example of thumbnails in the revision list area.

以降，各領域が保持する機能について詳述する．

3.1.1 リビジョン一覧領域

コントローラ領域で Save，Compile，Run のいずれかのボタンが押されたとき，リビジョン一覧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**）．図 4 にリビジョン一覧表示機能の例を示す．この領域では，四角の枠線で囲まれたサムネイルを用いてユーザがいつ，どのような操作を行ったのかを表現する．サムネイルは左上，右上，下部の 3 つの領域に分かれている．左上にはリビジョン ID が表示され，下部にはそのリビジョンがデータベースへ保存されたときの時刻が表示される．右上には，コンパイルエラーや実行時エラーの有無が表示される．「○」の表示はプログラムが異常終了しなかったことのみを示しており，必ずしも実行時

に正しい結果が出力されたとは限らない．また，サムネイルの枠線には実線と点線があり，実線はそのリビジョンが差分表示領域に表示されていることを示している．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，赤色の色に対応している．図 4 の例では，ユーザが順に Save，Compile，Run の動作を行い，Compile を行ったときにコンパイルエラーが生じたが，その後の修正により Run を行ったときはエラーが生じなくなったことを示している．

3.1.2 変更内容一覧領域

変更内容一覧領域には 4 リビジョン分，過去と直近の，ソースコードと結果の差分を表示する領域がある（**差分表示機能**）．この各リビジョンのソースコードと出力の差分を表示する領域を差分表示領域と呼ぶ．ユーザが任意のリビジョンをリビジョン一覧領域から選択するか，ユーザの Save，Compile，Run の操作によって新たなリビジョンがデータベースへ保存されるたびに，変更内容一覧領域の右下に新たな差分表示領域が出現する．なお，差分表示領域は最大で 4 つまで表示され，古いものから順に左上，右上，左下，右下に配置されている．新しいリビジョンが右下へ追加されたとき，最も古い差分表示領域が削除され，再度整列される．

図 5 に差分表示領域の例を示す．この領域では，過去の特定の ID のリビジョンと最新のソースコードとの差分が表示される．特定の ID のリビジョンに対して，最新のソースコードで追加された文字は背景が緑色になり下線が引かれ，削除した文字は背景が赤色になり打ち消し線が引かれる．差分表示領域の Source code と Output という文字の左に表示されている目のアイコンをクリックすることで，元の出力にあたる Original，あるいは差分表示にあたる Diff に表示を切り替えることができる（**表示切替え機能**）．

また，Pockets ではソースコードをコンパイルあるいは実行したときに得られる出力結果を，以下の 3 種類に分類している．

Output1 標準出力
Output2 コンパイルエラー出力
Output3 実行時エラー出力

差分出力領域に存在する元のリビジョンの出力と，最新のリビジョンの出力結果の種類が同じであった場合，初期状態として差分が表示される．差分を表示することでユーザによるソースコードへの変更とその変更にともなう出力結果の変移が明らかになり，ユーザが複数種類の実装を試行・評価する探索的プログラミングを容易に行うことが可能となると考えられる．一方，過去のリビジョンと最新のリビジョンで出力の種類が異なる場合，出力結果の差分は元の出力結果にあたる Output(Original) が表示され，目のアイコンを押しても差分は表示されなくなる．これは，出力結果の種類が違うものどうしの差分を表示して，ユーザが混乱することを防ぐためである．

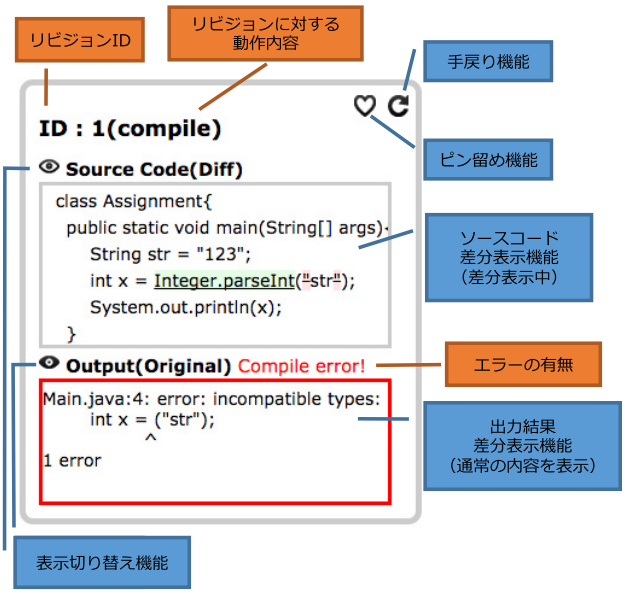

図 5 差分表示領域の表示例
Fig. 5 Example of differences view area.

各差分表示領域の右上にある矢印アイコンをクリックすることで，クリックした過去の特定のリビジョンのソースコードがエディタ領域に読み込まれる（**手戻り機能**）．この機能により，探索的プログラミング中における過去のソースコードへの手戻りが容易となる．また，差分表示領域に表示されていないリビジョンに対しても，サムネイルをクリックすることで手戻りを行うことができる．

また，特定のリビジョンを差分表示領域に表示し続けておきたい場合は，各リビジョン表示領域右上のハートマークを選択する（**ピン留め機能**）．ピン留めされた状態のリビジョンはユーザが変更の基点にしているとし，新しいリビジョンとの入れ替えが行われない．

3.2 Pockets の実装

Pockets のサーバの OS は Linux で，Web サーバおよびデータベースサーバはそれぞれ Apache，MySQL を使用している．実装言語および実装フレームワークは，サーバサイドが PHP および ZendFramework である．3.1 節で述べた各機能は，クライアントサイドにおいて Javascript および jQuery を使用して実装している．また，エディタ領域でコーディングされたソースコードのコンパイルおよび実行は，オンラインコンパイラである Ideone [*5] の API を使用している．

4. ケーススタディ

初学者が探索的プログラミングを行う際に Pockets がどのように役に立つかを調査するために 2 つのケーススタディを行った．ケーススタディ 1 では，大阪府立大学工業

[*5] http://ideone.com/

高等専門学校における 41 名の本科 2 年生の学生を対象とした C 言語プログラミング演習授業に Pockets を演習環境として提供し，学生が探索的プログラミングを行ううえで Pockets をどのように利用しているか調査した．ケーススタディ 2 では，奈良先端科学技術大学院大学の修士 10 名の学生を対象として，実験を行った．実験では，Pockets を使用した場合と使用していない場合において，探索的プログラミングの試行回数や課題の回答にかかった時間を比較することで，より詳細な Pockets を使用した探索的プログラミングの特徴に関して調査を行った．

4.1 ケーススタディ 1

ケーススタディ 1 の概要を述べる．ケーススタディ 1 は，大阪府立大学工業高等専門学校本科 2 年生 41 名が受講している，C 言語の習得を目的とした科目の第 13 回目に実施した．なお，本科目は合計 15 回の授業から構成されている．本実験の目的は，Pockets を用いて探索的プログラミングにおけるソースコードの複雑化，それによる混乱を防ぐことだけでなく，Pockets が実際の初学者向けプログラミング演習でも使用可能であるかを調査するためである．受講生らは，この回までに C 言語の基礎的な文法，条件分岐，繰返し構文，配列について学習済みである．また，当該科目はプログラミングの導入科目であるため，独学により事前にプログラミングを学習していた学生を除いた多くの学生はプログラミング経験が半年未満である．このことから，今回対象とした学生は Pockets が対象としているプログラミング初学者にあてはまるといえる．

実験の手順としては，講義開始時に Pockets の操作方法について説明を行い，その後 45 分間で Pockets を使用して演習課題のコーディングを行ってもらった．演習課題として難易度が違う問題を 4 問与え，演習時間内で問題をできる限り解くように指示した．問題を解く順序に関しては特に指定していない．なお，これらの問題はすべて，学生がすでに学習済みのプログラミング知識を用いて解くことができる．

4.1.1 評価方法

ケーススタディ 1 では，学生が Pockets の各機能をどのように使用したのかを調査するため，なお，演習中は教員 1 名，教員補助 1 名が課題の意味やエラー解消の方法が分からない学生の指導など，演習の補助を行ったが Pockets の操作を促すような指導は行っていない．Pockets により取得されたデータの分析と，実験終了後に Pockets のユーザビリティに関するアンケート調査を行った．

データの分析は，Pockets により取得されたデータのうち，2.2 節で定義した狭義の探索的プログラミングにあてはまる箇所を著者らが目視で抽出した．また，2.2 節で定義した狭義の探索的プログラミングの定義で述べている「複数回の変更」については，より多くのデータを収集するために「2 回以上の連続した変更」である場合，探索的プログラミングを行っているものと判断した．

4.1.2 実験結果

今回観測された探索的プログラミングの事例は 16 件で，うち 8 件が手戻りをともなうものであった．また，手戻りをともなった探索的プログラミングのうち，6 件は Pockets の手戻り機能を使用していたことを確認した．また，手戻り機能が用いられていなかった 2 件についてどのように手戻りを行っていたのかを分析したところ，2 件とも 1 行程度の細かな手戻りであり，3 分以内に保存されたソースコードへと戻るものであった．このことから，Pockets の手戻り機能を使う必要がないと学生が判断したと考えられる．さらに，手戻り機能が利用された 6 件についても同様に分析を行ったところ，コンパイルエラーが連続した場合や本来不要な記述をソースコードに追加してしまった場合に利用されていた．また，手戻り先のソースコードの状態を確認したところ，必ずしも解答と一致した出力ではなくても，最初に出力が成功したリビジョンや連続していたコンパイルエラーが解消されたリビジョンに戻っていることが観測された．以上より Pockets の手戻り機能は，学生による試行錯誤が行き詰まった際に，1 度分かりやすい状態に戻る目的で利用されることが確認できた．これは，初学者は Pockets を使用することによって，探索的プログラミング中に行われる手戻りや細かな修正によるソースコードの複雑化を防ぐことが可能になると考えられる．

4.2 ケーススタディ 2

ケーススタディ 2 の概要を述べる．ケーススタディ 2 では，より詳細に Pockets 特有の機能が探索的プログラミングをどのように促進するかを調査することを目的とした．ケーススタディ 2 の被験者は 4 名が修士 1 年生，6 名が修士 2 年生であるが，全員 C 言語の授業を受けた経験は 2 年以下である．また実験後アンケートにおいて，10 名中 8 名は日常的に C 言語を使用していないと述べ，残りの 2 名についても授業外で使用することもあるが頻繁ではないと述べている．したがって，本論文では初学者を「高等教育機関における特定のプログラミング言語を対象をした学習経験が 2 年以下のもの」と定義し，ケーススタディ 2 の被験者らを C 言語に関する初学者として取り扱う．

Pockets 特有の機能が探索的プログラミングに及ぼす効果を調査するため，本実験では被験者 10 名を 5 名ずつのグループに分け，それぞれ Pockets を使用した場合と使用しなかった場合で課題を 1 問ずつ解いてもらい，そのデータを収集した．各グループの各課題における使用ツールは**表 1** に示したとおりである．課題の内容を付録 A.1，A.2 へ示す．課題は両方とも条件分岐，Loop 文，配列のみで解くことが可能であり，使用言語は C 言語を指定した．また，実験時間の都合上，それぞれの課題の回答時間として

表 1 グループ概要
Table 1 Outline for group on case study 2.

項目	内容	
グループ	A	B
人数	5 名	5 名
使用ツール（課題 1）	Pockets	C3PV
使用ツール（課題 2）	C3PV	Pockets

45 分の制限を設けた．課題 1 と課題 2 で使用ツールおよびその順番を変更した理由は，ツールの慣れによる各機能の操作数に偏りが生じないようにするためである．

3 章で述べたように，C3PV は Pockets のベースとなったシステムであり，C3PV を用いてコーディングする際は，図 3 に示す「C3PV オリジナル領域」のみがブラウザに表示される．収集されるデータについては，Pockets で拡張した差分表示領域に表示されているリビジョン，手戻りを行ったリビジョンおよびピン留めされたリビジョン以外のデータが C3PV では収集される．

4.2.1 評価方法

我々は Pockets を使用することによって得られる効果や利点について，以下の観点を基準として設けた．そして以下の項目において，Pockets のすべての機能を使用したときと，C3PV オリジナル領域のみでコーディングを行ったとき，どのような差が出るか調査を行った．

観点 1 手戻りをともなう探索的プログラミングの回数
観点 2 手戻りをともなわない探索的プログラミングの回数
観点 3 保存，コンパイル，実行，手戻り（Save，Compile，Run，Revert）の回数
観点 4 課題を解くのにかかった時間とその達成度

観点 4 について，課題 1 は被験者全員が指定した時間内に完答することができなかったため，付録 A.1.1 のようなサブゴールを 7 つ設けた．そして，被験者らのコーディング過程から，どの程度各項目を達成しているかで達成度の評価を行った．

4.2.2 実験結果

観点 1，観点 2 の結果を**表 2** に，観点 3 の結果を**表 3** に，観点 4 の結果を**表 4** にそれぞれ示す．表 2 の括弧内の数値は，確認された手戻りのうち Pockets の手戻り機能を使用して手戻りを行っている回数である．表 3 の括弧内の数値は，Save あるいは Run により表示されたリビジョンのうち，ピン留めされたリビジョンの数である．表 4 の達成度は，付録 A.1.1 で詳述した課題 1 におけるサブゴールを満たした個数である．また，被験者 B は時間内に課題 2 を回答することができなかったため，解答時間は評価の対象外とする．

表 2 より，手戻りをともなう探索的プログラミングの総回数は課題 1，2 ともに Pockets のほうが C3PV より増加

表 2 手戻りとツールの関係
Table 2 The relation of backtracking and tool.

	手戻りあり		手戻りなし	
	課題 1	課題 2	課題 1	課題 2
Pockets	2(2)	4(1)	4	1
C3PV	0	1	4	0

表 3 ケーススタディー 2 動作内訳
Table 3 Detail of action for case study 2.

課題	課題 1		課題 2	
グループ	A	B	A	B
ツール	Pockets	C3PV	C3PV	Pockets
Save	33(2)	56	19	17
Compile	28	56	27	16
Run	40	46	56	58(4)
Revert	4	-	-	1
合計	105	158	102	92

表 4 達成度（課題 1）と解答時間（課題 2）
Table 4 Achievement degree and solved time.

グループ	A					B				
被験者	A	B	C	D	E	F	G	H	I	J
課題 1 達成度（個）	5	3	4	2	4	4	5	3	3	5
課題 2 解答時間（分）	15	–	14	16	15	17	15	19	14	14

したことが確認された．また，表 2，3 の結果より，課題 2 では Pockets の手戻り機能が 4 回の探索的プログラミングのうち 1 回しか利用されていなかった．そこで手戻り機能を使用していない 3 回の手戻りについて詳細を確認したところ，3 回すべてにおいて手戻り先が差分表示領域に表示されており，被験者がそれを見ながら手戻り先リビジョンへ，手入力により手戻りを行っていたことが推測される．以上より，差分表示領域によって過去のソースコードと出力結果を閲覧できるようになったことで，被験者が特定のソースコードに加えた変更や実行結果を振り返りやすくなったと考えられる．実際に，被験者によって利用された Pockets のピン留め機能の履歴を確認したところ，エラーが生じなかったリビジョンや，エラーが発生する直前のリビジョンがピン留めされており，被験者にとって分かりやすい過去の状態を把握する目的で差分表示領域が用いられていたことが分かった．以上より，今回の実験においては多くの被験者が，Pockets の差分表示領域や手戻り機能によって複数種類の実装の試行・評価が容易になり，探索的プログラミングをより積極的に行うようになったと考えられる．

5. 考察

本論文において我々は 2.3 節で述べた課題の改善を目的として Pockets を提案した．ケーススタディ 1 においては，コンパイルエラーや実行時エラーが解消されたリビ

ジョンに戻る目的で Pockets の手戻り機能が利用されていたことが確認された．このときの手戻り先が必ずしも解答と一致するものではないことからも，探索的プログラミングにおけるソースコードの複雑化にともなう混乱を避け，学生にとって分かりやすい状態に手戻りをする手段として，Pockets が利用されていたと考えられる．ケーススタディ 2 では，Pockets の手戻り機能や差分表示機能を利用した探索的プログラミングがより積極的に行われるようになっていることが観測できた．以上より Pockets は初学者による探索的プログラミング実施時における混乱を回避する手段として有用であるといえる．

ケーススタディ 1 において，学生らにとって Pockets の提供する機能やインタフェースがどの程度適切であったかを定性的に評価するために，袴田ら [14] の手法に準じたアンケートを実施した．袴田らは，システムが適切な機能・インタフェースであれば，ユーザは強制しなくても自然に利用するという仮説を立証している．アンケート項目とその結果を以下に示す．

質問 1 本日使用したツールに関して，良かったと思う点を選択してください．（選択式，複数選択可）

- **選択項目 1** 過去の Save，Compile，Run の結果がサムネイルとして一覧で見ることができること（リビジョン一覧表示機能）
- **選択項目 2** 過去に書いたソースコードに簡単に戻ることができること（手戻り機能）
- **選択項目 3** 過去のソースコードと現在のソースコードの差分を見ることができること（差分表示機能）
- **選択項目 4** 特になし
- **選択項目 5** その他（自由記述）

質問 2 本日使用したツールに関して，使いにくかったと思う機能や改善点を教えてください．（自由記述）

質問 3 ツールを使用したことによって，普段プログラミング演習において使用しているプログラミング環境に比べ，課題は解きやすくなったと思いますか？（「思う」または「思わない」を選択）

質問 4 上記で「思う」もしくは「思わない」を選んだ理由を教えてください．（自由記述）

ケーススタディ 1 の質問 1 から 3 の結果をまとめた表を**表 5** に示す．なお，質問 2 は自由記述であったため，分析者が各項目に分類した．ここで，分類項目 “その他” として，オンラインエディタのバグやツール全体の文字の小ささ，ウィンドウサイズを変更した際のレイアウトの崩れなどが指摘されていた．

質問 1「良かったと思う機能」において，“特になし” と回答した 3 名を除いた 35 名から Pockets の機能のうち少なくとも 1 つは良かったという回答を得た．さらに，質問 3「ツールを使用したことにより普段のプログラミング環境より課題が解きやすくなったと感じるか」については，38

表 5 ケーススタディ 1 結果（有効回答数 38 件）
Table 5 Result of case study 1.

質問 1：良かったと思う機能	
リビジョン一覧表示機能	26 件
手戻り機能	22 件
差分表示機能	20 件
特になし	3 件
質問 2：不便だと思った機能	
リビジョン一覧表示機能	1 件
手戻り機能	2 件
差分表示機能	10 件
その他	9 件
特になし	13 件
質問 3：普段のプログラミング環境に比べ課題は解きやすくなったと思うか	
思う	28 件
思わない	10 件

名中 28 名が「思う」と回答した．その理由として（質問 4），「差分表示機能により過去の実行結果が閲覧可能だったため」，「リビジョン一覧表示機能で表示されるのエラーの有無が○と×で簡単で分かりやすかったため」，「手戻り機能により初期状態へ容易に戻ることが可能だったため」と，Pockets 固有の機能に対する意見を多く得た．なお，質問 3 で課題が解きやすくなったと回答した 28 名に対して詳細な回答を記述してもらったところ，リビジョン一覧表示機能，手戻り機能など Pockets 固有の機能について言及した学生は 17 名，オンラインエディタの機能について言及した学生が 7 名，両機能について言及した学生が 4 名となった．つまり，合計 21 名が Pockets 固有の機能により課題が解きやすくなったと回答している．実際に Pockets の手戻り機能を使用した事例は 6 件であったが，22 名が手戻り機能を良かったと思うと回答している．この理由として，ケーススタディ 1 において実験前に行った事前講習が関係していると考えられる．すべての学生には事前講習において差分表示機能，リビジョン一覧表示機能，手戻り機能を使用したときの，Pockets の動作や表示内容について確認してもらっている．さらに質問 1 では「良かったと思う点を選択してください」と聞いているため，今回の実験では手戻り機能を使用していない学生でも，手戻り機能に関して良いと感じた場合投票していると考えられる．実際に，自由記述で得られた回答を確認したところ，「過去に書いたソースコードに戻れるのが間違えてしまったときに便利だなと思った」という回答が存在した．

一方で質問 2 において，38 名中 10 名が差分表示機能を不便だった機能としてあげた．不便に感じた理由について 10 名へ詳細な回答を記述してもらった結果，コンパイルエラーが生じたときに差分表示領域の出力覧が小さすぎること，エラー出力どうしの差分の表示が分かりにくいためという回答が得られた．学生のプログラミング演習に対する

モチベーション低下を防ぐためにも，今後，出力部分の拡大やレイアウトなどの改善を図っていきたい．

また，ケーススタディ 2 でも実験後アンケートとして，ケーススタディ 1 のアンケートにおける質問 1 とプログラミングの経験年数を被験者に対して質問したところ，10 名中 8 名が差分表示機能，手戻り機能，サムネイル機能のいずれかについて使いやすいと思うと回答した．その理由として差分表示機能では「加えた変更と結果がまとまっていることで，手戻り先のリビジョンを容易に選ぶことができたから」，「前のリビジョンでなぜエラーが生じたのか，差分が表示されているのですぐ分かったため」，手戻り機能では「初期化してしまいたいときに容易に最初のリビジョンへ戻ることが可能であるため」，サムネイル一覧機能では「過去の成功したリビジョンがすぐ分かるため」という回答を得た．これらの意見からも，Pockets の差分表示機能は探索的プログラミングにおける手戻りの難しさを緩和し，手戻り失敗によるソースコードの複雑化を防ぐことができると考えられる．

これらのアンケート結果より，多くの初学者が Pockets の機能やインタフェースを有用であると感じていること，実際に Pockets の機能を利用した探索的プログラミングが初学者によって行われていたことが分かった．以上より，不慣れな言語におけるプログラミングの課題において，Pockets が初学者による探索的プログラミング，すなわち複数種類の実装を試行・評価しながら開発を進めていくことを支援する効果があることが確認できた．

一方で今回のケーススタディ 2 の結果から分かるように，Pockets の利用は必ずしも課題の達成度や解答時間の改善にはつながっていない．これは，プログラミング課題を解く際には，Pockets が支援する探索的プログラミングだけでなく，課題の内容やアルゴリズムを理解する能力，複数のライブラリやアルゴリズムを組み合わせて実際にソースコードを完成させる能力といった様々な能力が求められることが原因であると考えられる．今後，初学者のプログラミングにおける思考モデルの分析などを通じて，より広範囲にわたるプログラミング支援環境の構築を目指していきたい．

6. 関連研究

Scratch [*6] に代表されるビジュアルプログラミング環境は初学者が探索的にプログラミングを進めることを前提として開発されている．このような環境では．あらかじめ選択肢にあたるブロックが複数用意されている．ユーザはこれらのブロックから特定のブロックを選び，ブロックどうしをつないでいくことでプログラムが自身の意図した挙動を示すようにコーディングを行うことができる．これは Brandt ら [5] が確認した探索的プログラミングのプロセスと一致する．しかし，ビジュアルプログラミングは Java や C といった言語を対象とした初学者向けプログラミング演習には向いておらず，プログラミング未経験者がプログラミングに慣れるための教材として通常利用されている．

Myers ら [9] や Yoon ら [10] は探索的プログラミングにおける手戻りに着目した研究を行っている．彼らは，ソフトウェア開発において開発者らはどの程度の頻度で手戻りを行っているか，また手戻りを行う理由はなぜなのかについて調査を行った．さらに，彼らは調査の結果から，既存の IDE は開発者が求めている手戻り機能を実装していないと指摘し，手戻りの種類（ソースコードの 1 カ所のみ手戻りを行う，あるいは複数箇所同時に手戻りを行うなど）が選択可能である Eclipse のプラグイン，AZURITE [*7] を開発している．AZURITE により，開発者は様々な種類の手戻りを行うことが可能となる．AZURITE は初学者による利用を想定していないため，初学者向けプログラミング演習にそのまま適用することが可能であるかは判断できないが，手戻りの支援によって Pockets と同様探索的プログラミングが促進される可能性がある．一方で，我々の先行研究 [8] において，初学者は探索的プログラミングを行う際，こまめに修正を行い，そのつどコンパイルや実行を行っていることが分かっている．また，初学者はエラーが生じるソースコードのまま探索を続けたり，1 度エラーが生じたソースコードを再度コンパイルあるいは実行を行ったりしていたため，熟練の開発者に比べてエラーが生じる回数が多いと予測される．そのため手戻りを行う際には，我々が実施したケーススタディによるアンケート結果でも言及があったように，Pockets の機能の一部である過去のソースコードの実行結果や現在との差分提示がより有用であると思われる．

また，編集履歴を開発者に提示するものとして，Github [*8] などが存在する．しかし，Github は版管理システム（Git）を使用していることが前提であるため，初学者にとって操作することは容易ではない．さらに，これらのシステムの可視化はファイル単位であるため，探索的プログラミングのような特定の 1 行や変数に着目した変更履歴の可視化には適切ではない．

7. おわりに

本研究では，初学者による探索的プログラミングに着目し，探索的プログラミングの促進を目的としたプログラミング環境 Pockets を提案し，その実装を行った．我々が実施した 2 回のケーススタディにより，初学者は Pockets のリビジョン一覧表示機能や手戻り機能を使用することで，探索的プログラミングにおけるソースコードの複雑化に

*6 https://scratch.mit.edu/

*7 http://www.cs.cmu.edu/~azurite/

*8 https://github.com/

よる混乱を回避することが可能であることを確認した．また，Pockets の差分表示機能は，探索的プログラミングにおける複数種類の実装の試行・評価を容易にすることが分かった．

今後の課題として，被験者の一部に機能が不便であると指摘された差分表示機能について，より理解しやすい表示方法を検討する必要がある．さらに，課題の達成度や回答時間の改善につなげるためには，課題の内容を理解する能力や，既知のライブラリやアルゴリズムに対する能力など，様々な能力が必要となる．今後は，初学者のプログラミングにおける思考モデルの分析などを通じ，初学者のプログラミング能力の向上へつなげていきたい．

謝辞 大阪府立大学工業高等専門学校本科 2 年生および奈良先端科学技術大学院大学情報科学研究科の実験にご協力していただいた学生の皆様へ深く感謝いたします．

参考文献

[1] 榊原康友，松澤芳昭，酒井三四郎：プログラミング初学者におけるコンパイルエラー修正時間とその増減速度の分析，情報教育シンポジウム 2012 論文集，Vol.2012, No.4, pp.121–128 (2012).
[2] Vihavainen, A., Luukkainen, M. and Kurhila, J.: Using Students' Programming Behavior to Predict Success in an Introductory Mathematics Course, *Proc. 6th International Conference on Educational Data Mining*, pp.300–303 (2013).
[3] Sheil, B.: Environments for exploratory programming, *Datamation*, Vol.29, No.7, pp.131–144 (1983).
[4] Rosson, M.B. and Carroll, J.M.: The Reuse of Uses in Smalltalk Programming, *ACM Trans. Computer-Human Interaction*, Vol.3, No.3, pp.219–253 (1996).
[5] Brandt, J., Guo, P.J., Lewenstein, J., Dontcheva, M. and Klemmer, S.R.: Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code, *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pp.1589–1598 (2009).
[6] Sandberg, D.W.: Smalltalk and exploratory programming, *ACM SIGPLAN Notices*, Vol.23, pp.85–92 (1988).
[7] Carnegie Mellon University: Variations to Support Exploratory Programming, available from ⟨http://www.exploratoryprogramming.org/⟩.
[8] 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).
[9] Myers, B.A., Oney, S., Yoon, Y. and Brandt, J.: Creativity Support in Authoring and Backtracking, *Proc. Workshop on Evaluation Methods for Creativity Support Environments at CHI*, pp.40–43 (2013).
[10] Yoon, Y. and Myers, B.A.: An Exploratory Study of Backtracking Strategies Used by Developers, *Proc. CHASE*, pp.138–144 (2012).
[11] 槇原絵里奈，藤原賢二，Uthayopas, P.，Chantrapornchai, C.，Fakcharoenphol, J.，井垣 宏，吉田則裕，飯田 元：日本とタイにおけるプログラミング初学者のプログラミング行動の比較，電子情報通信学会技術研究報告，Vol.114, No.260, pp.46–51 (2012).
[12] Tamada, H., Ogino, A. and Ueda, H.: A Framework for Programming Process Measurement and Compile Error Interpretation for Novice Programmers, *Proc. Joint Conference of the 21th International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pp.233–238 (2011).
[13] 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).
[14] 袴田大貴，松澤芳昭，太田 剛：初学者向けデバッガ DENO の利用実態の分析，電子情報通信学会技術研究報告，Vol.123, No.5, pp.1–8 (2014).

付 録

A.1 課題 1 出題内容

課題 1：数字の螺旋

入力された値を N（N は 2 以上 9 以下の自然数）としたとき，N×N 個の数値を，1 から順番に螺旋状に表示するプログラムを作成せよ．数値は左上端を基点に，下，右，上，左…と，外側から内側に進むにつれ増えていく．

出力例 (N = 9 のとき)：
1 32 31 30 29 28 27 26 25
2 33 56 55 54 53 52 51 24
3 34 57 72 71 70 69 50 23
4 35 58 73 80 79 68 49 22
5 36 59 74 81 78 67 48 21
6 37 60 75 76 77 66 47 20
7 38 61 62 63 64 65 46 19
8 39 40 41 42 43 44 45 18
9 10 11 12 13 14 15 16 17

出力例 (N = 2 のとき)：
1 4
2 3

A.1.1 課題 1 サブゴール

SG1 表示する内容を記録するための配列を作っている．
SG2 表示する内容を記録するための配列を初期化している．
SG3 表示する内容を記録するための配列を使っている．
SG4 ループ中に数値を埋める進行方向を変更するようなプログラムを作成しようとしている．
SG5 行単位で出力する値を決定しようとしている．
SG6 多重ループを使用して配列に数値を格納している．
SG7 配列の出力を行っている．

A.1.2 サブゴールの設定方法

課題 1 を解くにあたって，我々は以下のモデル解法を設

定した．

手順 1 数値を格納するための配列を作成する（SG1，SG2 が対応）．

手順 2 数値を格納するためのアルゴリズムを作成する．

手順 3 出力用の二次元配列へ数値を格納する（SG3 が対応）．

手順 4 手順 3 で格納した配列の中身を出力する（SG7 が対応）．

ここで，手順 2 における数値を格納するためのアルゴリズムは様々なものが考えられるため，第 2 著者と相談し，初学者が行う可能性のあるアルゴリズムとして以下の 3 つを設けた．

- ループと二次元配列を用いて，下，右，上，左…のように，螺旋状に数値をインクリメント，あるいはデクリメントしながら二次元配列へ格納するアルゴリズム（SG4 が対応）．
- 行単位で格納する数値を計算．出力していくアルゴリズム（SG5 が対応）．
- 多重ループと条件分岐を用いて数値を格納するアルゴリズム（SG6 が対応）．

本実験においては，望ましい出力が得られなかった場合でも上記アルゴリズムを使用した時点でサブゴールを達成したものと判断している．

A.2 課題 2 出題内容

課題 2：Sum of 4 Integers

50 以下の正の整数 n を入力し，0〜9 の範囲の整数 a，b，c，d の組で

$a + b + c + d = n$

を満たすものの組合せ数を出力するプログラムを作成せよ．

たとえば，n が 35 のとき，(a，b，c，d) は (8，9，9，9)，(9，8，9，9)，(9，9，8，9)，(9，9，9，8) の 4 通りなので，答えは 4 となる．

槇原 絵里奈 （学生会員）

平成 25 年大阪工業大学情報科学部情報システム学科卒業．平成 27 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在，同大学院博士後期課程に在学．修士（工学）．ソフトウェア工学教育，特にプログラミング教育支援に興味を持つ．

藤原 賢二 （正会員）

平成 22 年大阪府立工業高等専門学校総合工学システム専攻修了．平成 27 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（工学）．現在，同大学院博士研究員．リファクタリングの適用履歴分析，プログラミング教育支援に関する研究に従事．

井垣 宏 （正会員）

平成 12 年神戸大学工学部電気電子工学科卒業．平成 17 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学院特任助手．平成 27 年大阪工業大学情報科学部准教授．博士（工学）．ソフトウェア工学教育，サービス指向アーキテクチャ，ソフトウェアプロセス等の研究に従事．

吉田 則裕 （正会員）

平成 16 年九州工業大学情報工学部知能情報工学科卒業．平成 21 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員（PD）．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．平成 26 年より名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

飯田 元 （正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 7 年奈良先端科学技術大学院大学情報科学センター助教授．平成 17 年同大学情報科学研究科教授．博士（工学）．