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.

自動採点システムを用いた講義における無作為修正者の提出行動分析

三野　天羽†　　上野　秀剛††

† 奈良工業高等専門学校システム創成工学専攻情報システムコース
†† 奈良工業高等専門学校情報工学科
E-mail: †ai1030@nara.kosen-ac.jp, ††uwano@info.nara-k.ac.jp

あらまし　大学等のプログラミング教育では，少数の教員が多数の受講者に対して講義を行うことが多く，受講者それぞれの様子や理解状況の把握は困難である．本研究では自動採点システムを用いた講義において，誤答の原因を考察せずに修正する行動の検出を目的とする．受講者を連続した提出の有無と，ソースコード提出時に集計した無作為修正の自己申告の有無で分類し，提出回数や課題の正答に対する行き詰まりを表すメトリクスの差を比較した結果，短時間に連続してソースコードを提出する受講者は課題に行き詰まっている可能性が高く，その編集履歴から無作為修正者である事が示唆された．

キーワード　プログラミング教育，Online Judge System，無作為修正

An Analysis of Random Corrector's Submission Activity on Programming Exercise using Automatic Scoring System

Takaha MINO† and Hidetake UWANO††

† Dept. Information Engineering, National Institute of Technology, Nara College
†† Dept. Information Engineering, National Institute of Technology, Nara College
E-mail: †ai1030@nara.kosen-ac.jp, ††uwano@info.nara-k.ac.jp

1. はじめに

大学等のプログラミング教育では，受講者が与えられた課題に対してソースコードを作成し提出を行うプログラミング演習と呼ばれる授業が開講されている．プログラミング演習において受講者の理解度には差があるため，教員は課題に行き詰まっている受講者に対して指導を行う必要がある．しかし，少数の教員が多数の受講者に対して講義を行うことが多いため，受講者それぞれの様子や理解状況を把握することは困難である．

講義に用いられるシステムの 1 つに, Online Judge System(OJS) がある．OJS は受講者が課題に対してソースコードを作成し提出すると，自動でコンパイルと実行を行う．実行の結果である標準出力は教員が用意した正解と比較することで採点され，採点結果が受講者に提示される．受講者は採点結果をもとにソースコードの修正と提出を繰り返し，正しい出力をするソースコードを実装する．この提出行動から理解度など受講者の状況識別を行うことが出来れば，課題に行き詰まっている受講者を特定して指導することが可能になる．

本研究では OJS を用いた講義において，教員による指導が必要な受講者のソースコードの提出行動にどのような特徴があるかを分析する．本稿では，教員による指導が必要な受講者のうち無作為修正者を対象とする．無作為修正者は，プログラミング演習において誤答の原因を考察せずに修正する受講者のことで，そのような受講者は課題に行き詰まっている可能性が高い．また，修正する際にシステムの採点結果から正否を確認し，自らで正否の判断を行わないため，短い時間に連続した提出を行う傾向があると考えられる．そこで本稿では受講者を，連続した提出の有無による分類と，ソースコード提出時に集計した無作為修正の自己申告による分類を行い，提出回数と課題の行き詰まりを表すメトリクスとの関係を分析する．無作為修正者の識別が可能になれば，課題に行き詰まっている受講者を特定して指導し理解を促すことができると考えられる．

2. 関連研究

槇原らは学生が講義中に行う探索的プログラミング行動を検出し，学生が課題にどのように取り組み，どのような箇所で

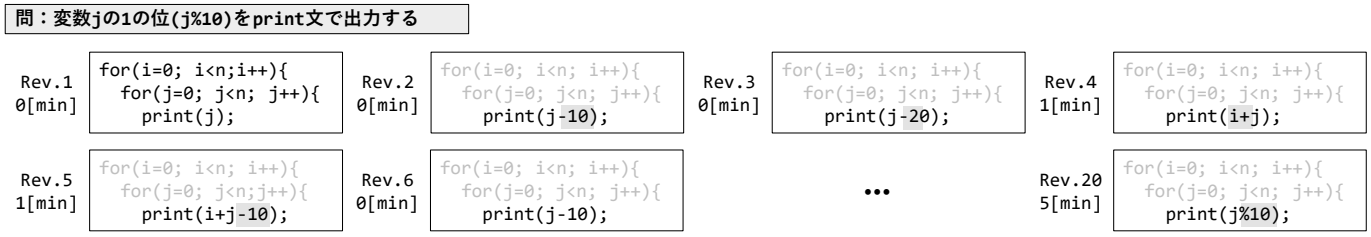

図 1　無作為修正の例

行き詰まっているのかをリアルタイムで特定する手法を提案した [1]．探索的プログラミングとは，実装が不明確な箇所に対して修正・コンパイル・実行・結果の確認を繰り返し行うことを指す．そのような行動をする学生は，特定の機能の実装につまずいている可能性がある．提案手法を実際のプログラミング演習に適用したところ，同一課題における学生間のアプローチの違いや，エラーが生じた原因の特定が容易になることが確認できた．

藤原らはプログラミング演習時に提出されたソースコードのスナップショットを分析することで，受講生がいつどのような箇所で行き詰まっていたのかを特定する手法を提案した [2]．提案手法は，受講生のある時点におけるソースコードと最終提出のソースコードの差分から，ある時点での推定残作業量を算出する．そして，推定残作業量が減少していない時間帯を検出し，その時間帯のスナップショットから行き詰まり箇所を特定し，講師に提示する．提案手法を実際のプログラミング演習において収集したスナップショットに適用することで，37 名の受講者のスナップショットから 46 件の行き詰まり箇所を特定することができた．

大野らはこれまでの研究において受講生が提出したソースコードのスナップショットから無作為修正を行う受講者を識別するメトリクスを提案した [3]．提案手法は，「同じ箇所を頻繁に修正する」，「一定時間の修正・コンパイル回数が多い」，「1 回あたりの修正量が小さい」という無作為修正者に特有の性質をメトリクスとして算出する．演習や講義の成績との相関を分析した結果，ソースコードの修正行が偏っていて提出回数が多い受講者の点数が低く，その編集履歴から無作為修正者である事が示唆された．

これらの研究はいずれも行き詰まりや無作為修正など，ソースコードの修正に進捗が見られない点に着目した分析を行っている．本研究ではメトリクスに基づいた行き詰まりの評価を行う際に，受講者にアンケートを採ることで実際に無作為な修正を行っている受講者の特徴を明らかにする．

3. 準　　備

3.1 OJS を用いたプログラミング講義

本研究が対象とするプログラミング講義は，ある単元に対する座学の後に演習課題を解く形式の講義である．また，受講者は演習課題で作成したソースコードを OJS に提出し即座に採点結果を得る．OJS は提出されたソースコードをコンパイル・実行し，教員があらかじめ作成したテストケースと出力結果が一致するか判定する．テストケースとは作成したプログラムが要件を満たしているかテストするための入力と，入力に対して望まれる出力の組みを表す．テストケースと出力結果の一致数に応じて以下の式で点数 *score* が計算される．

課題	受講者	区間								
		0-5	5-10	10-15	15-20	20-25	25-30	30-35	35-40	40-45
1	A		x							
1	B	xox			xx					
1	C		?o							
2	A	x		?						
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

アンケート結果
無作為：o
考えた：x
無回答：?

図 2　5 分ごとに分割した際の提出の様子

$$score = \frac{\text{テストケースの正解数}}{\text{テストケース数}} \times 100 \tag{1}$$

受講者には採点結果として各テストケースの入力と正誤判定結果，*score*，実行時エラー，コンパイル時エラーなどが提示される．受講者は *score* が 100 点になるまで，ソースコードの修正と提出を繰り返す．作成されたソースコードは提出ごとにリビジョンとして，提出日時，点数とともに記録される．

3.2 無作為修正者

本研究ではプログラミング演習において誤答の原因を考察せずにソースコードの修正を繰り返す受講者を「無作為修正者」と定義する．図 1 に無作為修正の例を示す．灰色の文字は前回の提出から差分のない行を，背景色が灰色の部分は前回の提出からの修正箇所を表す．また，*Rev.N* は N 回目の提出，*M*[*min*] は *Rev.*1 提出からの経過時間を表す．この例ではリビジョン 1 の提出から 5 分間で 20 回の提出をしているが，すべてのリビジョンで 3 行目の print 文のみを修正している．また，リビジョン 2 とリビジョン 6 で同じ内容を提出しており，修正内容について方略が見られない．無作為修正者は複数のリビジョンをかけて，if 文や for 文の条件式，print 文の内容といった，特定の行のみを対象に，軽微な修正を繰り返す傾向がある．

プログラミング演習において，作成したソースコードにエラーや不具合を発見した受講者は，その原因を考察・理解した上でソースコードを修正することが望ましい．しかし，無作為修正者はエラーや不具合の原因を考えず，自分が持っている修正パターンを闇雲に適用することで正解にたどり着こうとすることが考えられる．そのため，闇雲な修正によってプログラムが正しく動いた場合，与えられた課題の仕様やアルゴリズム，新たに学習した文法などを理解せずに演習を終え

る可能性がある．無作為修正者の識別が可能になれば，課題に行き詰まっている受講者を特定し指導することで理解を促すことができると考えられる．

3.3 メトリクス

本稿では，提出回数，一定時間ごとの提出回数，行き詰まり率の 3 つのメトリクスとアンケート結果の関係を分析する．

提出回数は 1 人の受講者がある課題に対してソースコードを提出した回数を表す．OJS を用いた講義では課題に対応するソースコードを提出すると採点結果が得られる．無作為修正者は誤答の原因と思われる箇所に対して闇雲に修正を行い，システムに繰り返し提出することで採点結果から正否を判断するため，提出回数が多くなると考えられる．

一定時間ごとの提出回数は各受講者が課題に対する 1 回目のソースコード提出を行った日時を基準として，最終版の提出までを一定時間の区間に分割したときの各区間におけるソースコードの提出回数を表す．本稿では分割する時間を 5 分とする．図 2 に 5 分ごとに分割した際の提出の様子を示す．図は各受講者が課題に対する 1 回目のプログラム提出を行った日時を 0 分とした，5 分ごとの提出を表す．記号の種類はソースコードの提出時に行うアンケートに対する回答結果を表す．アンケートについては 4.1 節で説明する．一定時間ごとの提出回数は図の各セル内の記号の数であり，値が高いほど短時間に連続してソースコードを提出していることを表す．無作為修正者は誤答の原因を考えずにシステムの採点結果から正否を確認しようとするため，短い時間に連続した提出を行う傾向があると考えられる．そのため一定時間ごとの提出回数が多くなると考えられる．

行き詰まり率は 1 人の受講者のある課題に対する 2 回目以降のソースコード提出の内，誤答の原因となっている誤りを 1 つも修正できない提出の割合を表す．誤答の原因が複数ある場合，1 つの誤りを修正しても *score* が変化しない場合がある．本稿では *score* の変動に関わらず，正答に近づくような修正があれば行き詰まりとは見なさない．行き詰まり率は値が高いほど正しい修正方法を理解しないままソースコードを修正・提出していることを表す．無作為修正者は修正の正否について自分では判断しないため，誤答の原因を解消できない提出が多くなり，行き詰まり率が高くなると考えられる．

4. 実　　験

4.1 環　　境

奈良工業高等専門学校の 3 年生が受講するプログラミングⅡを対象とする．受講者数は 38 人である．講義は 1 回 90 分間で，ある単元に対して 30 分程度の説明を受けた後に残り時間で演習課題を解く形式である．1 回の講義で 1〜3 個の演習課題が出題され受講者はこれを Java で実装する．講義では OJS を用いて課題の出題，回収，採点を行っている．OJS にはオープンソースの OJS である SharifJudge を日本語化し，複数ファイルを提出可能とする機能拡張をしたシステムを使用している．また，本実験のために OJS にアンケート機能を追加している．

図 3　OJS の画面遷移

受講者がソースコード提出，アンケート回答，採点結果確認をする際の画面遷移を図 3 に示す．受講者は課題に対して作成したソースコードを提出すると，そのソースコードが無作為に修正または作成されたかを問うアンケート画面に遷移する．アンケートに回答すると採点結果表示画面に遷移する．アンケートは受講者が内容を読まずに回答する可能性を考え，既定値を無回答に設定する．しかし，採点結果を早く確認したい受講者がアンケートを選択せずに回答すると，アンケートの無回答率が高くなる．OJS ではソースコードの提出から採点完了までに 10〜20 秒時間がかかるため，ソースコードの提出直後に採点結果表示画面に遷移しても結果は表示されない．そこで，アンケート画面にジャッジ中であることを表示して，アンケート回答に充てる時間があることを認知させ，回答率が高くなるようにする．

4.2 分　　析

本稿では，4.1 節で説明した講義において収集した計 416 回 (5 課題, 延べ 182 人) の提出データを分析対象とする．

5 分ごとの提出回数は以下の手順で算出する．

（1） 課題 ID と受講者 ID から同じ受講者の同じ課題に対するソースコード提出の日時とアンケート結果を抽出する

（2） 受講者が課題に対する 1 回目のソースコード提出を行った日時を 0 分として，5 分ごとの区間に分割する（図 2）

（3） 各セル内の記号の数を求める

図 4 に各区間ごとの提出数を示す．区間 40-45 までの提出数が全提出数の 76 %を占めており，それ以降の区間における提出の大部分が前回提出から 1 時間以上空いているため，5 分ごとの提出回数では区間 0-5 から区間 40-45 を分析対象とする．また，セル内の記号の数が 0 の場合は分析対象に含めない．各セルを無作為修正の有無で分類し提出回数の平均値に差があるか検証する．

提出回数は，課題 ID と受講者 ID から同じ受講者が同じ課題に対してソースコードを提出した回数を求める．*score* が 100 に到達した後に提出がある場合は，誤答の修正ではなくリファクタリングであるとみなし，提出回数に含めない．

行き詰まり率は，同じ受講者が同じ課題に対して提出した

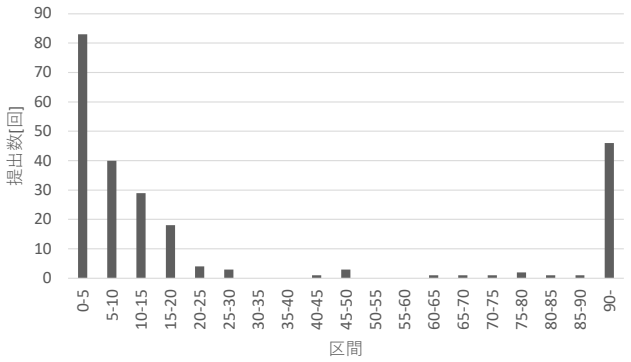

図 4　区間ごとの提出数

表 1　5 分ごとの提出回数

		無作為修正	
		あり	なし
頻度	1 回	10　(58.8%)	100　(81.3%)
	2 回	5　(29.4%)	17　(13.8%)
	3 回	2　(11.8%)	6　(4.9%)
提出回数の平均値		1.53	1.24
p 値		0.035	

ソースコードの変更履歴を元に，誤答の原因を修正できたか著者の 1 人が目視で確認し算出する．1 回目の提出で *score* が 100 に到達している場合は分析対象に含めない．

提出回数と行き詰まり率は，無作為修正の有無と連続提出の有無で群分けし平均値に差があるか検証する．連続提出の有無は，同じ受講者が同じ課題に対して行ったソースコード提出において，5 分ごとの提出回数が基準値を超える区間が存在する場合に「有」とする．基準値は 5.1 節の結果をもとに決定する．

5.　結果と考察

5.1　5 分ごとの提出回数

表 1 に無作為修正あり／なしそれぞれの 5 分ごとの提出回数の頻度と平均値，平均値に対する検定結果（Wilcoxon の順位和検定）を示す．無作為修正あり群の 5 分ごとの提出回数の平均値 (1.53) は無作為修正なし群 (1.24) より 0.29 高く，有意差 (p=0.035) が見られた．また，頻度を見ると無作為修正あり群は 5 分間に 2 回以上提出する割合が 41.2% であるのに対して，無作為修正なし群では 18.7% と大きな差が見られる．

結果は，1 章で述べたように無作為修正者がシステムの採点結果から修正の正否を確認するために，短い時間に連続した提出を行う傾向があることを示唆する．次節では，連続提出の基準を 5 分間に 2 回以上としたときの連続提出の有無による群間の差を分析する．

5.2　提出回数と行き詰まり率

表 2 に無作為修正の有無と連続提出の有無で分類した提出回数と行き詰まり率の平均値，および検定結果を示す．提出回数は Wilcoxon の順位和検定を，行き詰まり率は Welch の t 検定を行った．無作為修正なし群と全体（無作為修正の区分なし）

問：以下のプログラムにtry/catchを追加し，エラーメッセージの代わりに”Out of Bound!!”と表示するようにせよ．表示は範囲を超えたアクセスがあるたびに行うようにせよ．

```
import java.util.Scanner;

public class Main {
  public static void main(String[]args ) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    sc.close();

    String[] s = new String[3];

    for(int i=0;i  < n;i ++)
      s[i] = "No." +i ;

    for(int i=0;i  < n;i ++)
      System.out.println(s[i]);
  }
}
```

図 5　問　題　文

Rev.1 : 0[min]

```
try {
for(int i=0; i < n;i ++)
  s[i] = "No." +i ;
}catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Out of Bound!!");
}
try {
for(int i=0; i < n;i ++)
  System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Out of Bound!!");
}
```

Rev.2 : 14[min]

```
for(int i=0; i < n;i++)
  try {
  s[i] = "No." +i;
  }catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Out of Bound!!");
}

for(inti =0; i < n;i++)
  try {
  System.out.println(s[i]);
  }catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Out of Bound!!");
  }
```

図 6　無作為修正なし，連続提出なしの修正

において，連続提出なし群より連続提出あり群の提出回数と行き詰まり率の平均値が高く，全てにおいて有意差 (p<0.000) が見られた．無作為修正あり群においても，連続提出なし群より連続提出あり群の提出回数と行き詰まり率の平均値が高かったものの，いずれも有意差は見られなかった．また，連続提出なし群と全体（連続提出の区分なし）において，無作為修正なし群より無作為修正あり群の平均提出回数が高く，有意差 (p=0.001) が見られた．それ以外の群間においては有意差は見られなかった．

各群の受講者が同じ課題に対して提出した履歴から特徴を考察する．図 5 に対象とする課題の問題文を示す．受講者には問題文と，入力として 2,4,5 が与えられた場合に望まれる出力が提示されている．また，受講者は課題を解く前に例外処理と try/catch 文の使用方法について講義を受けている．

表 2　提出回数と行き詰まり率の平均値

		連続提出なし	連続提出あり	全体
無作為修正あり	平均提出回数	2.89(N=19)	3.86(N=7)	3.15(N=26)
		p=0.061		
	平均行き詰まり率 [%]	22.9(N=13)	42.6(N=7)	29.8(N=20)
		p=0.105		
無作為修正なし	平均提出回数	1.79(N=136)	4.5(N=20)	2.14(N=156)
		p<0.000		
	平均行き詰まり率 [%]	12.0(N=67)	40.4(N=19)	18.3(N=86)
		p<0.000		
全体	平均提出回数	1.93(N=155)	4.33(N=27)	2.29(N=182)
		p<0.000		
	平均行き詰まり率 [%]	13.7(N=80)	41.0(N=26)	20.4(N=106)
		p<0.000		

Rev.1 : 0[min]

```
  try{
  int n = sc.nextInt();
  sc.close();

  String[] s = new String[3];
  for (int i = 0; i < n; i++)
      s[i] = "No." +i ;
  for (int i = 0; i < n; i++)
      System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Out of Bound!!");
}
```

Rev.2 : 2[min]

```
  try{
  int n = sc.nextInt();
  sc.close();

  String[] s = new String[3];
  for (int i = 0; i < n; i++)
      s[i] = "No." + i;
  for (int i = 0; i < n; i++)
      System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Out of Bound‼");
}
```

Rev.3 : 5[min]

```
  int n = sc.nextInt();
  sc.close();


  String[] s = new String[3];
  try {
  for (int i = 0; i < n; i++)
    s[i] = "No." + i;
  for (int i = 0; i < n; i++)
    System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Out of Bound‼");
}
```

Rev.4 : 11[min]

```
  int n = sc.nextInt();
  sc.close();


  String[] s = new String[3];

  for (int i = 0; i < n; i++)
  try {
    s[i] = "No." +i ;
  }catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Out of Bound‼");
  }
  for (int i = 0; i < n; i++)
  try{System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Out of Bound‼");
}
```

Rev.5 : 12[min]

```
  int n = sc.nextInt();
  sc.close();


  String[] s = new String[3];

  for (int i = 0; i < n; i++)
  try {
    s[i] = "No." + i;
  }catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Out of Bound!!");
  }
  for (int i = 0; i < n; i++)
  try{System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e){
  System.out.println("Out of Bound!!");
}
```

図 7　無作為修正なし，連続提出ありの修正

無作為修正なし，連続提出なしの提出履歴を図 6 に示す．1 回目の提出は try ブロックの範囲が誤っており，14 分後の 2 回目の提出で正しい位置に修正し，正答しているため，行き詰まり率は 0%，提出回数は 2 回である．図が示すように無作為修正なし，連続提出なしに属する受講者は，誤答の原因を考察し時間をかけて修正を行うため提出回数と行き詰まり率が低くなったと考えられる．

無作為修正なし，連続提出ありの提出履歴を図 7 に示す．この受講者も 1 回目の提出では try ブロックの範囲が誤っていることが誤答の原因であるが，2 分の提出（2 回目）で誤答の原因ではない出力部を修正し，5 分の提出（3 回目）で try 文を移動しているが誤答の原因の解消には至っていない．その後，これまでの提出よりも時間をかけた 11 分の提出（4 回目）と，12 分の提出（5 回目）において，2 つの誤答（1 つは 2 回目の提出で追加された）を順次修正し，正答に達している．1 回目から 3 回目，および 4 回目から 5 回目の提出が連続提出であり，行き詰まり率は 50 % (2/4)，提出回数は 5 回である．図の受講者は 3 回目の提出までは誤答の原因を軽微な誤りと考え，短時間に連続した修正を行ったが，正答できず，その後時間をかけて考察することで正答に到達したと考えられる．

無作為修正あり，連続提出ありの提出履歴を図 8 に示す．この受講者はアンケートで 3 回目の提出以外は全て無作為修正であると回答している．3 回目は未回答であった．提出履歴を見ると，図 6 や図 7 と同様に 1 回目の提出で try ブロックの範囲が誤っている事と，出力部の誤りが誤答の原因である．4 分の提出（2 回目）と 5 分の提出（3 回目）が連続提出であるが，いずれも誤答の原因を解消できていない．誤答の原因を解消できたのは 5 回目の提出（10 分）と 6 回目の提出（15 分）で，行き詰まり率は 60 % (3/5)，提出回数は 6 回である．無作為修正あり，連続提出ありに属する受講者も短時間に連続した提

Rev.1 : 0[min]

```
try {
int n = sc.nextInt();
sc.close();
String[] s = new String[3];
for(int i=0; i< n; i++)
  s[i] = "No." +i ;

for(int i=0; i< n; i++)
  System.out.println(s[i]);

}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out Of Bound!!");
}
```

Rev.2 : 4[min]

```
int n = sc.nextInt();
sc.close();
String[] s = new String[3];
try {
for(inti =0; i< n; i++)
  s[i] = "No." + i;

for(inti =0; i< n; i++)
  System.out.println(s[i]);

}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out Of Bound!!");
}
```

Rev.3 : 5[min]

```
int n = sc.nextInt();
sc.close();
String[] s = new String[3];
try {
for(int i=0; i< n; i++)
  s[i] = "No." + i;
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out Of Bound!!");
}
for(int i=0; i< n; i++)
  System.out.println(s[i]);
```

Rev.4 : 8[min]

```
int n = sc.nextInt();
sc.close();
String[] s = new String[3];
try {
for(int i=0; i< n; i++)
  s[i] = "No." +i ;
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out Of Bound!!");
}
try {
for(int i=0; i< n; i++)
  System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out Of Bound!!");
}
```

Rev.5 : 10[min]

```
int n = sc.nextInt();
sc.close();
String[] s = new String[3];
try {
for(inti =0; i< n; i++)
  s[i] = "No." + i;
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out of Bound!!");
}
try {
for(inti =0; i< n; i++)
  System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out of Bound!!");
}
```

Rev.6 : 15[min]

```
int n = sc.nextInt();
sc.close();
String[] s = new String[3];

for(int i=0; i< n; i++)
  try {
  s[i] = "No." + i;
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out of Bound!!");
}

for(int i=0; i< n; i++)
  try {
  System.out.println(s[i]);
}catch(ArrayIndexOutOfBoundsException e) {
  System.out.println("Out of Bound!!");
}
```

図 8　無作為修正あり，連続提出ありの修正

出をしている間は誤答の原因を解消できておらず，処理構造（try 文の範囲）の誤りについては，その後の時間をかけた提出で正答に到達している．

本課題において無作為修正あり，連続提出なしに属する受講者は 2 人であった．どちらの受講者も 1 回目の提出で正解しており，無作為修正は見られなかった．無作為修正であると回答した理由は，無作為に作成したソースコードが偶然正解した場合や，作成したソースコードが正解である確証が持てなかった場合が考えられる．無作為修正あり，連続提出ありと比較して行き詰まり率が大幅に低くなったのはこの様な受講者が含まれるためだと考えられる．

また，他の課題に対して提出された無作為修正あり，連続提出なしに属する受講者の提出履歴を確認すると，広い範囲を誤った形で修正している様子が見られた．この様な修正は無作為修正であってもソースコードの変更に時間がかかるため連続提出ではないものの，提出回数は多くなったと考えられる．

6. おわりに

本稿では，プログラミング演習における無作為修正者のソースコードの提出行動にどのような特徴があるのかを分析した．受講者を連続した提出の有無と，ソースコード提出時に集計した無作為修正の自己申告の有無で分類し，提出回数や課題の正答に対する行き詰まりを表すメトリクスの差を比較した．奈良高専におけるプログラミング講義で収集したデータを対象とした実験の結果，短時間に連続してソースコードを提出する受講者は課題に行き詰まっている可能性が高く，その編集履歴から無作為修正者である事が示唆された．本稿の結果から，連続した提出の有無で課題に行き詰まっている受講者を特定し，指導することで理解を促すことができると考えられる．

一方で，連続した提出を行っていない受講者の中にも，アンケートで無作為修正を行ったと回答した受講者が見られた．提出履歴を確認すると，広い範囲を誤った形で修正している様子が見られた．この様な修正は無作為修正であってもソースコードの変更に時間がかかるため連続した提出を含まず，本稿で用いたメトリクスでは把握することができない．連続した修正以外の基準を用いた無作為修正者の検出は今後の課題である．

また，本稿では連続した提出の有無を 5 分ごとの提出回数を用いて判断した．しかし，提出時間が $Rev.1$: 0[min], $Rev.2$: 5[min], $Rev.3$: 6[min] となる場合に $Rev.2$ と $Rev.3$ の提出間隔が 1 分であるのにも関わらず，区間 0-5 と区間 5-10 に分断されてしまい連続提出として検出することができない．この様な連続提出を検出することも今後の課題である．

文　献

[1] 槇原 絵里奈, 井垣 宏, 吉田 則裕, 藤原 賢二, 飯田 元 : プログラミング演習における探索的プログラミング行動の自動検出手法の提案, コンピュータソフトウェア, Vol.35, No.1, pp.110-116 (2018).

[2] 藤原 賢二, 上村 恭平, 井垣 宏, 吉田 則裕, 伏田 享平, 玉田 春昭, 楠本 真二, 飯田 元 : スナップショットを用いたプログラミング演習における行き詰まり箇所の特定, コンピュータソフトウェア, Vol.35, No.1, pp3-13 (2018).

[3] 大野 優, 上野 秀剛, 内田 眞司 : ソースコードのスナップショットに基づいた無作為修正者の検出, 電子情報通信学会技術研究報告, Vol.118, No.214, pp.53-58 (2018).