システム創成工学専攻
情報システムコース
Department of Systems Innovation
Advanced Information System Course

令和３年度 専攻科特別研究論文

# 自動採点システムを用いたプログラミング講義における無作為修正者の提出行動分析

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

指導教員名　上野 秀剛 准教授
論文提出者名　三野 天羽

独立行政法人 国立高等専門学校機構
奈良工業高等専門学校 専攻科
National Institute of Technology, Nara College
Faculty of Advanced Engineering

# 自動採点システムを用いたプログラミング講義における無作為修正者の提出行動分析

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

三野 天羽
Takaha Mino

独立行政法人 国立高等専門学校機構
奈良工業高等専門学校 専攻科 システム創成工学専攻 情報システムコース
大和郡山市矢田町 22 番地（〒 639-1080）
National Institute of Technology, Nara College, Faculty of Advanced Engineering
22 Yata-cho, Yamatokoriyama, Nara 639-1080, Japan

***Abstract:*** In programming education at school, a small number of instructors often lecture to many students. For that reason, it is hard to grasp each student's level of understanding. The purpose of this study is to detect the behavior of students who correct their source code without considering the cause of their mistakes in programming exercises using an automatic scoring system. We classify students by two factors, 1) consecutive submissions and 2) the questionnaire asking they had random corrected their source code. The results showed that students who submit source code continuously in a short time are likely stuck in their exercises, and their editing history suggests that they are random correctors.

***Keywords:*** programming education, Online Judge System, random correction;

# 関連業績リスト

1. 三野 天羽, 上野 秀剛 : 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, 信学技報 教育工学研究会, Vol.121, No.232, pp.31–36 (2021).

# 目次

# 図目次

# 表目次

# 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 [1]．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，教員は理解度の低い受講者に対して優先的に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 [2]．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．本研究では，課題に取り組み始めてから正しい出力をするソースコードを実装，提出するまでの一連の行動を提出行動と呼ぶ．提出行動の特徴から受講者の状況を識別できれば，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明らかにする．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，闇雲な修正によってプログラムが正しく動いた場合，課題の内容を理解せずに講義を終えてしまう可能性がある．その様な受講者は，修正の正否をシステムの採点結果から確認し，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．そこで本研究では受講者に対して，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．課題に行き詰まっている受講者を特定できれば，優先的に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 [3]．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 [4]．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 [5]．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履歴に基づいた作業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

本研究では提出行動を表すメトリクスの算出が容易である OJS を用いたプログラミング講義を対象とする．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．

$$score = \frac{\text{テストケースの正解数}}{\text{テストケース数}} \times 100 \tag{3.1}$$

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 3.1 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

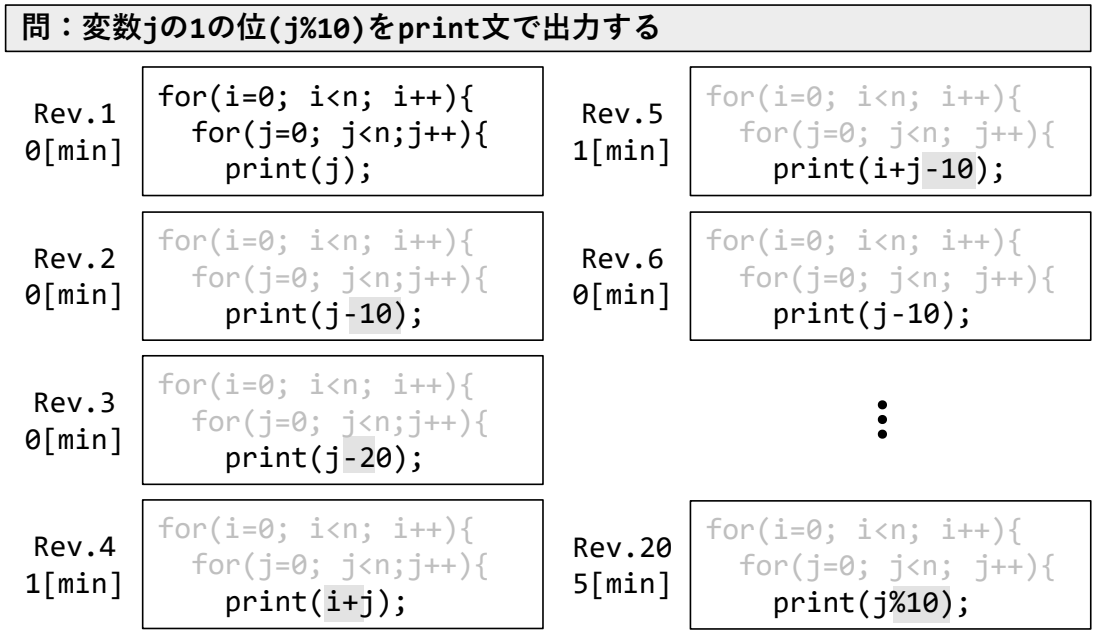


図 3.1　無作為修正の例

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．

## 3.3　メトリクス

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．本研究では分割する時間を 5 分とする．図 3.2 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．誤りが複数ある場合，1 つの誤りを修正しても *score* が変化しない場合がある．その場合は *score* の変動に関わらず，正答に近づく修正があれば誤りを修正できたとみなす．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．

| 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 3.2　5 分ごとに分割した際の提出の様子

# 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ソフトウェアの 1 つである 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．

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 4.1 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 秒の時間がかかるため，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．採点が終了すると表示は「ジャッジ中」から「ジャッジ完了」に変化する．


図 4.1　OJS の画面遷移

## 4.2　分析

4.1 節で説明した講義において収集した計 416 回 (5 演習課題, 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. 各提出に対して，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からの経過時間を求める
3. 5 分ごとに分割された各区間に対して，区間 A-B に経過時間が A 分を超えて B 分以下の提出に対するアンケート結果を割り振る（図 3.2）
4. 各区間に相当する時間中のソースコード提出の回数を数える

図 4.2 に分析対象の全ての提出における，各区間ごとの提出数を示す．図より 30 分までに多くの提出が行われており，それ以降に提出を行う受講者は他の受講者と性質が異なると考えられる．そのため，5 分ごとの提出回数では区間を絞って分析を行う．本研究で対象とする講義の講義時間の半分である 45 分までの提出を対象としても，全提出数の 76 % を占めることから区間 0-5 から区間 40-45 を分析対象とする．記号を 1 つ以上含むセルを無作為修正の有無で分類し，各セルの提出回数の平均値に差があるか検証する．

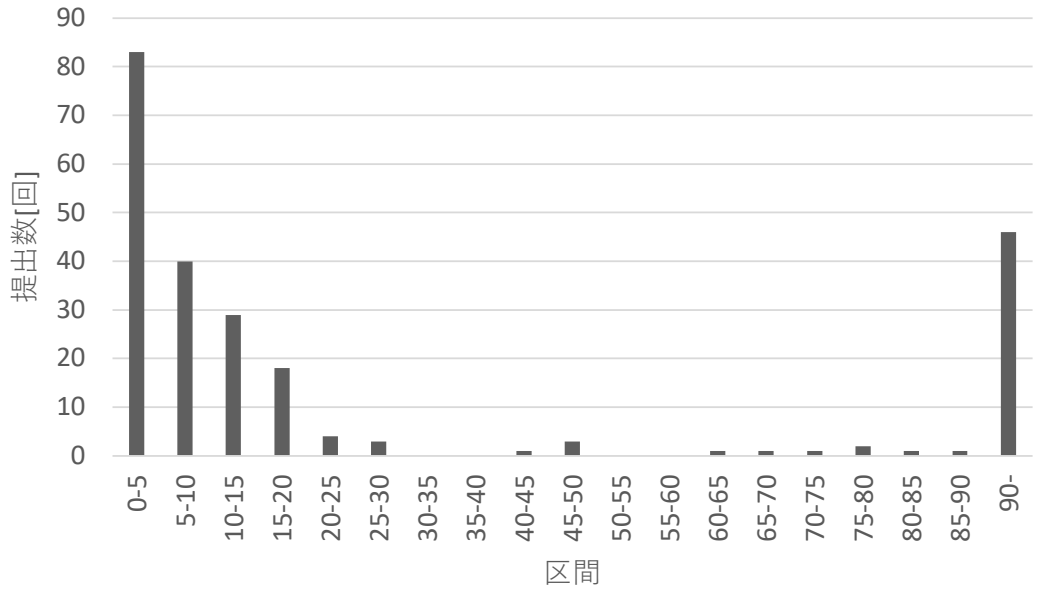


図 4.2　区間ごとの提出数

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」とし，1 つも存在しない場合に「無」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 5.1 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．

結果は無作為修正者が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．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であると考えられる．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．

表 5.1　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 | |

## 5.2 提出回数と行き詰まり率

表 5.2 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．また，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の箱ひげ図を図 5.1 に，行き詰まり率の箱ひげ図を図 5.2 に示す．

表 5.2 より，全体（連続提出の分類なし）において，無作為修正あり群の提出回数の平均値は無作為修正なし群より高く，有意差 (p=0.001) が見られた．無作為修正者は採点結果から修正の正否を確認するため提出回数が増えるからだと考えられる．また，行き詰まり率の平均値も無作為修正あり群が無作為修正なし群より高かったものの，有意差は見られなかった．

表 5.2 より，連続提出なし群においても，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，有意差が見られた．図 5.1 より，無作為修正なし群は提出回数が 2 回以下の範囲に多く分布しているが，無作為修正あり群は 2 回以上の範囲にも分布している．また，行き詰まり率の平均値も無作為修正あり群が無作為修正なし群より高かったものの，有意差は見られなかった．図 5.2 より，無作為修正なし群は行き詰まり率の分布が 0[%] に集中しているが，無作為修正あり群は 0[%] から 40[%] の範囲に分布している．

一方で表 5.2 より，連続提出あり群においては，無作為修正なし群より無作為修正あり群の提出回数の平均値が低かった（有意差なし）．図 5.1 より，無作為修正なし群と無作為修正あり群で分布にあまり差が見られない．また，無作為修正あり群の行き詰まり率の平均値 (42.6%) は無作為修正なし群 (40.4%) より 2.2 ポイント高いが，全体（連続提出の分類なし）における無作為修正あり群の行き詰まり率の平均値 (28.4%) と無作為修正なし群 (18.5%) の差 (9.9 ポイント) より小さかった．図 5.2 より，こちらも無作為修正なし群と無作為修正あり群で分布にあまり差が見られない．よって，連続提出あり群は全体（連続提出の分類なし）と連続提出なし群とは傾向が異なると考えられる．

表 5.2　提出回数と行き詰まり率の平均値

| | | 無作為修正なし | 無作為修正あり | 全体 |
|---|---|---|---|---|
| 連続提出あり | 平均提出回数 [回] | 4.50(N=20) | 3.86(N=7) | 4.33(N=27) |
| | | p=0.314 | | |
| | 平均行き詰まり率 [%] | 40.4(N=19) | 42.6(N=7) | 41.0(N=26) |
| | | p=0.827 | | |
| 連続提出なし | 平均提出回数 [回] | 1.79(N=136) | 2.89(N=19) | 1.93(N=155) |
| | | p<0.005 | | |
| | 平均行き詰まり率 [%] | 12.2(N=66) | 21.2(N=14) | 13.7(N=80) |
| | | p=0.253 | | |
| 全体 | 平均提出回数 [回] | 2.14(N=156) | 3.15(N=26) | 2.29(N=182) |
| | | p=0.001 | | |
| | 平均行き詰まり率 [%] | 18.5(N=85) | 28.4(N=21) | 20.4(N=106) |
| | | p=0.139 | | |

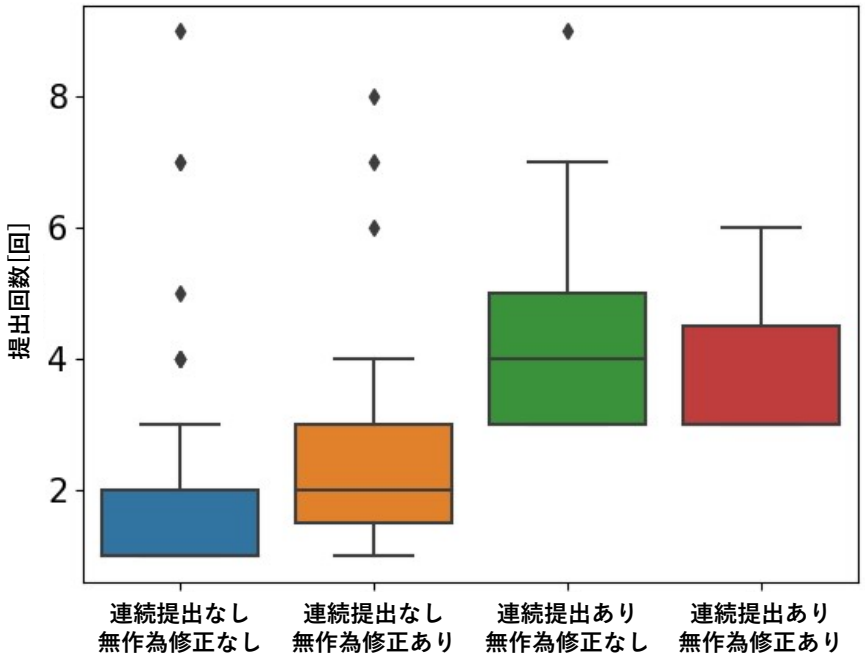


図 5.1　提出回数の分布

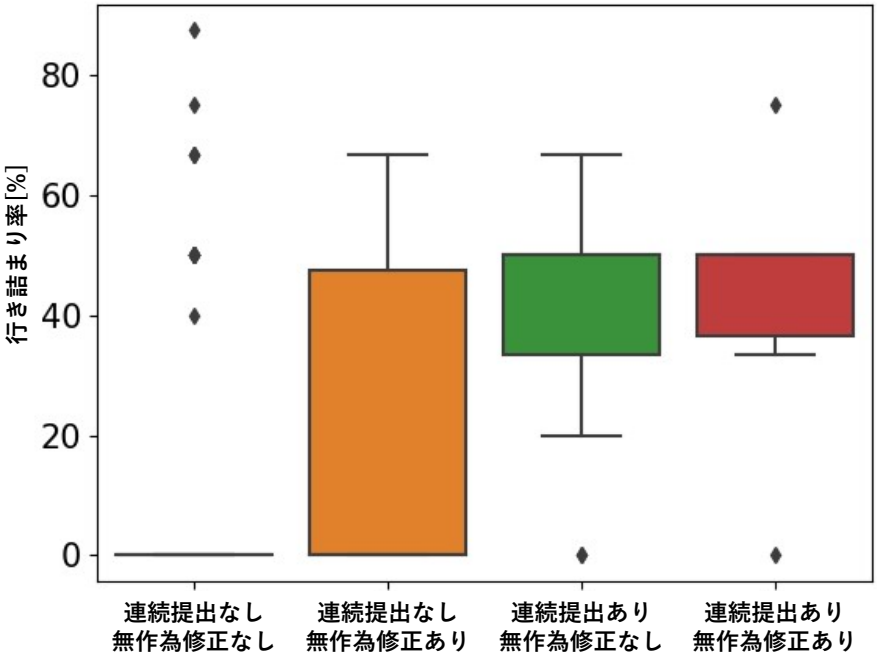


図 5.2　行き詰まり率の分布

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 5.3 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．

**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 5.3　問題文

無作為修正なし，連続提出なしに属する受講者 A の提出履歴を図 5.4 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．図の受講者は，誤答の原因を考察し時間をかけて修正を行った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．

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，連続提出なしの行き詰まり率 (21.2%) が無作為修正あり，連続提出あり (42.6%) と比較して大幅に低い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．広い範囲を誤った形で修正する受講者は，連続した提出以外の基準を用いて検

**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 5.4　無作為修正なし，連続提出なしの修正

出する必要がある．

無作為修正あり，連続提出ありの受講者 B の提出履歴を図 5.5 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，1 回目の提出で try ブロックの範囲が誤っている事（原因 1）と，出力部の誤り (原因 2) 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．その後，原因 2 を 5 回目の提出（10 分）で，原因 1 を 6 回目の提出（15 分）でそれぞれ解消しており，行き詰まり率は 60 % (3/5)，提出回数は 6 回である．図の受講者は，短時間に連続した提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，その後時間をかけて考察することで解消できたと考えられる．

**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 5.5　無作為修正あり，連続提出ありの修正

無作為修正なし，連続提出ありの受講者 C の提出履歴を図 5.6 に示す．1 回目の提出で try ブロックの範囲が誤っていることが誤答の原因（原因 1）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誤った形で修正し，出力部に誤り（原因 2）を追加してしまっている．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）で原因 1 を修正し，12 分の提出（5 回目）で原因 2 を修正し，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．図の受講者も，無作為修正あり，連続提出ありに属する受講者と同様に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．

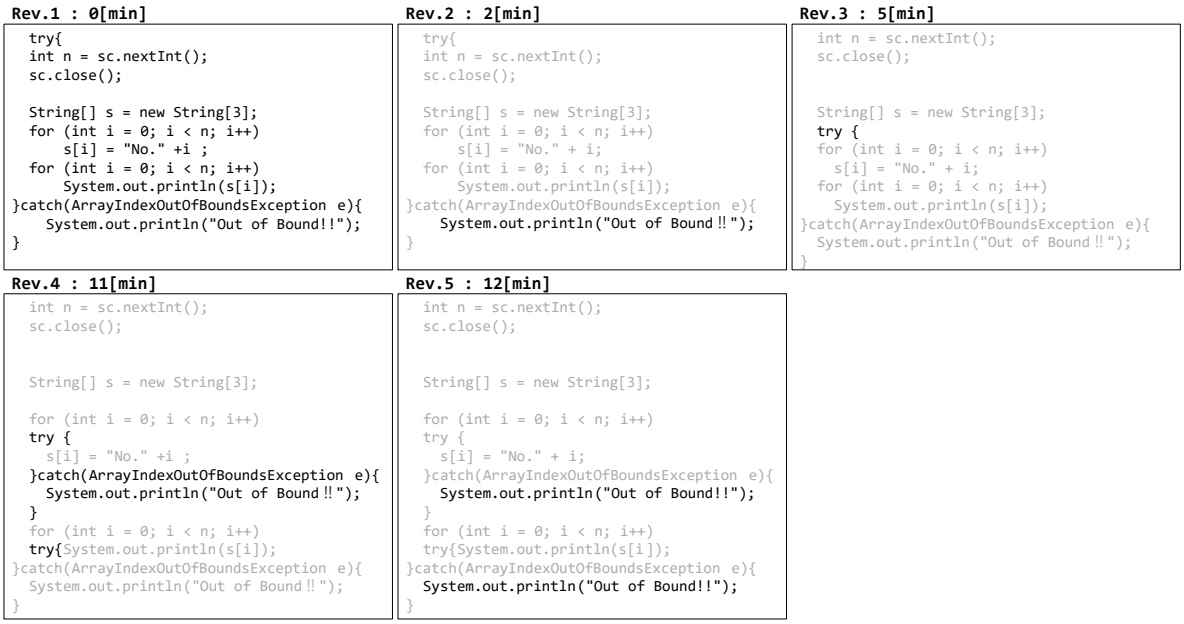


図 5.6　無作為修正なし，連続提出ありの修正

無作為修正なしと回答した受講者でも連続提出を含む場合に，無作為修正あり，連続提出ありの受講者と同じような修正を行う例が見られた．連続提出あり群の提出回数と行き詰まり率の平均値の傾向が，全体（連続提出の分類なし）と連続提出なし群とは異なるのは，この様な受講者が含まれるためだと考えられる．表 5.3 に連続提出の有無のみ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．連続提出あり群の提出回数と行き詰まり率の平均値は連続提出なし群より高く，いずれも有意差 ($p<0.000$) が見られた．この結果は無作為修正の有無で分類した場合と同じ傾向を示しており，連続した提出の有無を用いることで，アンケートとは異なる方法で課題に行き詰まっている受講者を特定することができると考えられる．

## 5.3　追加実験

5.2 節では，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

表 5.3　提出回数と行き詰まり率の平均値（連続提出による比較）

| | 連続提出なし | 連続提出あり |
|---|---|---|
| 平均提出回数 | 1.93(N=155) | 4.33(N=27) |
| | p<0.000 | |
| 平均行き詰まり率 [%] | 13.7(N=80) | 41.0(N=26) |
| | p<0.000 | |

| 区間 | 0-1 | 1-2 | 2-3 | 3-4 | 4-5 | 5-6 | 6-7 | 7-8 | 8-9 | 9-10 | … |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 提出の様子 | | | | | x | o | | | | | … |
| 移動平均 | | | | | 0.2 | 0.4 | 0.4 | 0.4 | 0.4 | 0.2 | … |

**アンケート結果**
**無作為：o**
**考えた：x**
**無回答：?**

図 5.7　提出回数の 5 分移動平均

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ために，提出回数の 5 分移動平均というメトリクスを新たに定義する．

提出回数の 5 分移動平均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 1 分の区間に分割し，区間数 5 で移動平均を取った提出回数を表す．図 5.7 に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，1 分ごとの提出の様子と移動平均を表す．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．移動平均は直近 5 区間のセル内の記号の数の平均値で，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．

### 5.3.1　分析

提出回数の 5 分移動平均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. 各提出に対して，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からの経過時間を求める

3. 1 分ごとに分割された各区間に対して，区間 A-B に経過時間が A 分を超えて B 分以下の提出に対するアンケート結果を割り振る（図 5.7 提出の様子）
4. 各区間に相当する時間中のソースコード提出の回数を数える
5. 区間 4-5 以降の各区間で直近 5 区間の提出回数の平均値を求める（図 5.7 移動平均）

4.2 節と同様の理由で提出回数の 5 分移動平均では区間 0-1 から区間 44-45 を分析対象とする．また，移動平均の算出結果が 0 の場合は分析対象に含めない．各区間を無作為修正の有無で分類し移動平均の平均値に差があるか検証する．

提出回数と行き詰まり率は 4.2 節と同様の手順で算出し，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，提出回数の 5 分移動平均が基準値を超える区間が存在する場合に「有」とし，1 つも存在しない場合に「無」とする．

### 5.3.2 結果と考察

表 5.4 に無作為修正あり／なしそれぞれの提出回数の 5 分移動平均の頻度と平均値，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提出回数の 5 分移動平均の平均値 (0.32) は無作為修正なし群 (0.26) より 0.06 高く，有意差 (p<0.000) が見られた．また，頻度を見ると無作為修正あり群は移動平均が 0.4 以上，つまり 5 分間に 2 回以上提出する割合が 46.7% であるのに対して，無作為修正なし群では 23.7% と大きな差が見られる．

表 5.4　提出回数の 5 分移動平均

| | | 無作為修正 | |
|---|---|---|---|
| | | あり | なし |
| 頻度 | 0.2 | 40 (53.3%) | 455 (76.3%) |
| | 0.4 | 27 (36.0%) | 112 (18.8%) |
| | 0.6 | 7 (9.3%) | 23 (3.9%) |
| | 0.8 | 1 (1.3%) | 6 (1.0%) |
| 移動平均の平均値 | | 0.32 | 0.26 |
| p 値 | | <0.000 | |

平均値と頻度の割合のいずれも 5 分ごとの提出回数と同様の結果を示しており，連続提出の有無を表すメトリクスとして使用できると考えられる．以降では連続提出の基準を提出回数の 5 分移動平均が 0.4 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．

表 5.5 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．また，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の箱ひげ図を図 5.8 に，行き詰まり率の箱ひげ図を図 5.9 に示す．

全体（連続提出の分類なし）は，表 5.2 と同じ結果である．

表 5.5 より，連続提出なし群において，無作為修正あり群の提出回数と行き詰まり率の平均値が無作為修正なし群より高かったものの，有意差は見られなかった．図 5.8 より，無作為修正なし群は提出回数が 2 回以下の範囲に多く分布しているが，無作為修正あり群は 2 回以上の範囲にも分布している．また，図 5.9 より，無作為修正なし群は行き詰まり率の分布が 0[%] に集中しているが，無作為修正あり群は 0[%] から 40[%] の範囲に分布している．

一方で表 5.5 より，連続提出あり群においては，無作為修正なし群より無作為修正あり群の提出回数の平均値が低かった（有意差なし）．図 5.8 より，無作為修正なし群と無作為修正あり群で分布にあまり差が見られない．また，無作為修正あり群の行き詰まり率の平均値 (33.8%) は無作為修正なし群 (37.4%) より 3.6 ポイント高いが，全体（連続提出の分類なし）における無作為修正あり群の行き詰まり率の平均値 (28.4%) と無作為修正なし群 (18.5%) の差 (9.9 ポイント) より小さかった．図 5.9 より，無作為修正なし群は行き詰まり率の分布が 30[%] から 50[%] に集中しているが，無作為修正あり群は 0[%] から 50[%] の範囲に分布しており，連続提出なし群の分布とは異なる結果が見られた．

表 5.5　提出回数と行き詰まり率の平均値

| | | 無作為修正なし | 無作為修正あり | 全体 |
|---|---|---|---|---|
| 連続提出あり | 平均提出回数 [回] | 4.19(N=26) | 3.92(N=12) | 4.11(N=38) |
| | | p=0.327 | | |
| | 平均行き詰まり率 [%] | 37.4(N=25) | 33.8(N=12) | 36.2(N=37) |
| | | p=0.692 | | |
| 連続提出なし | 平均提出回数 [回] | 1.73(N=130) | 2.50(N=14) | 1.81(N=144) |
| | | p=0.101 | | |
| | 平均行き詰まり率 [%] | 10.1(N=60) | 21.1(N=9) | 12.0(N=69) |
| | | p=0.287 | | |
| 全体 | 平均提出回数 [回] | 2.14(N=156) | 3.15(N=26) | 2.29(N=182) |
| | | p=0.001 | | |
| | 平均行き詰まり率 [%] | 18.5(N=85) | 28.4(N=21) | 20.4(N=106) |
| | | p=0.139 | | |

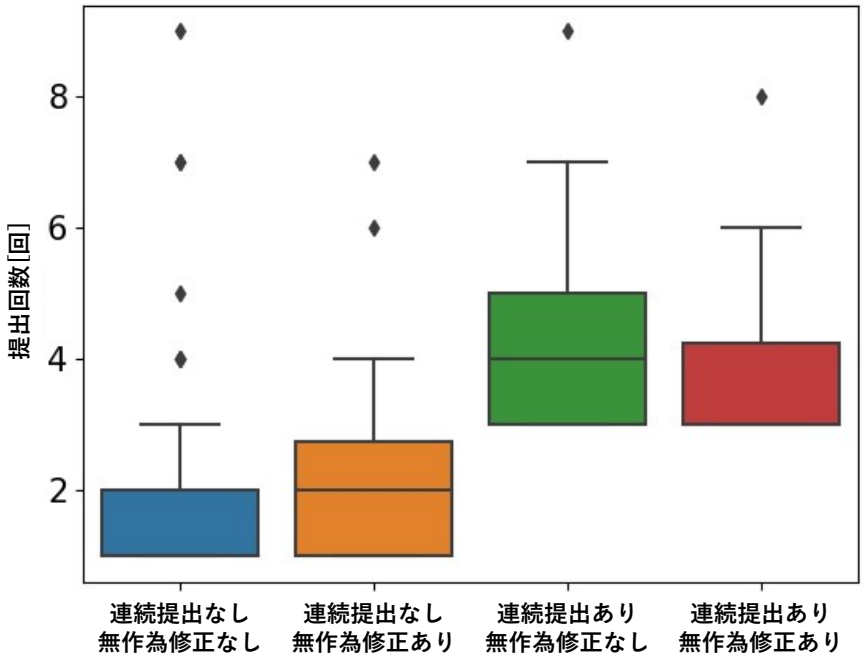


図 5.8　提出回数の分布

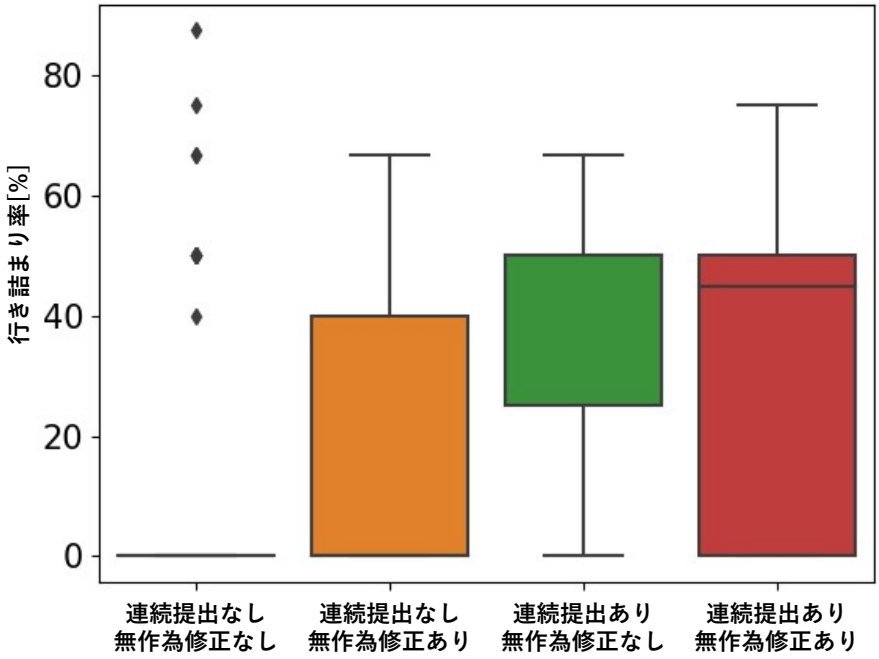


図 5.9　行き詰まり率の分布

表 5.6 に連続提出の有無のみ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．連続提出あり群の提出回数と行き詰まり率の平均値は連続提出なし群より高く，いずれも有意差 ($p<0.000$) が見られた．

表 5.5, 5.6 は 5 分ごとの提出回数を用いて分類した場合と同様の結果を示しており，提出回数の 5 分移動平均による分類が有効であると考える．また，5 分ごとの提出回数を用いて連続提出ありと判断した件数は提出回数が 27 件，行き詰まり率が 26 件であった（表

表 5.6　提出回数と行き詰まり率の平均値（連続提出による比較）

| | 連続提出なし | 連続提出あり |
|---|---|---|
| 平均提出回数 | 1.81(N=144) | 4.11(N=38) |
| | p<0.000 | |
| 平均行き詰まり率 [%] | 12.0(N=69) | 36.2(N=37) |
| | p<0.000 | |

表 5.7　提出回数の 5 分移動平均で検出可能となった連続提出

| 無作為修正 | 提出回数 [回] | 行き詰まり率 [%] |
|---|---|---|
| なし | 3 | 0.00 |
| なし | 4 | 66.7 |
| なし | 3 | 50.0 |
| なし | 3 | 0.00 |
| なし | 3 | 0.00 |
| なし | 3 | 50.0 |
| あり | 3 | 0.00 |
| あり | 8 | 57.1 |
| あり | 3 | 0.00 |
| あり | 3 | 50.0 |
| あり | 3 | 0.00 |

5.3）のに対して，提出回数の 5 分移動平均を用いた場合は提出回数が 38 件，行き詰まり率が 37 件（表 5.6）と増加した．

表 5.7 に増加した 11 件の無作為修正の有無，提出回数，行き詰まり率を示す．11 件中 5 件で無作為修正が含まれていた．また無作為修正を含まない 6 件中 3 件で，誤答の誤りを 1 つも修正できない提出が含まれていた．上記の結果から，5 分ごとの提出回数では検出できない連続提出を提出回数の 5 分移動平均を用いて検出できることが確認できた．これにより，より高い精度で課題に行き詰まっている受講者を特定することができると考えられる．

# 6. おわりに

本研究では，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．

本研究の結果から，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，本研究で用いたメトリクスでは把握することができない．連続した提出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．

本研究では連続した提出の有無を，5 分ごとの提出回数と提出回数の 5 分移動平均がしきい値を超えるか否かで判断した．しかし課題の難易度や受講者全体の理解度合いにより，しきい値は変化するため本研究で示したしきい値を他の講義でも適用することは難しい．メトリクスを波形として分析し，無作為修正者の典型的な波形を算出し分類ができれば，波形の一致度を用いてしきい値に依らない分類が可能になると考えられる．

また，本研究では移動平均の幅を 5 分としたが，幅を 10, 30 分と長くして無作為修正者の長期的な特徴を把握することは興味深い発展の 1 つだと考える．

# 謝辞

本研究を進めるにあたり，多くの方々のご助力をいただきました．この場を借りてお礼を申し上げます．

指導教員である上野秀剛准教授には，研究を進めるにあたってご指導いただきました．心から感謝申し上げます．

また，査読教員である市川嘉裕助教をはじめ，内田眞司教授，山口賢一教授，本間啓道准教授，松村寿枝教授には，貴重な意見や指摘をいただきました．誠に感謝しております．

# 参考文献


[1] 独立行政法人情報処理推進機構 : IT 人材白書 2017, 独立行政法人情報処理推進機構 (2017).

[2] 松永 賢次 : 導入プログラミング教育におけるオンラインジャッジシステムの活用の試み, 情報科学研究, Vol.31, pp.25–41 (2010).

[3] 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).

[4] 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).

[5] 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).