Vol.2017-CE-138 No.4
2017/2/11

初学者向けプログラミング授業における活動ログの評価支援機能

長 慎也[1,a)]　長島 和平[2]　間辺 広樹[3]　兼宗 進[4]　並木 美太郎[2]

概要：
筆者らは現在，Web 上でプログラミング学習が可能な環境「BitArrow」を，初学者向けのプログラミング学習活動に用いている．BitArrow にはログ取得機能があり．授業担当者からの希望があった場合には，利用者がプログラムを保存するごとにプログラムの内容などを記録している．ログをもとに利用者のプログラミングの理解度を測りたい．理解度を測るためには，最終的なプログラムの成否（出力が正しいどうか）だけではなく，そのプログラムの中でプログラムを構成する要素（変数や制御構造など）をどの程度応用できたか，また最終的な成果物にたどり着くまでの過程をも評価することが重要である．しかし，ログに蓄積されたプログラムの量は膨大であり，すべてのプログラムについて評価するのは手間がかかる．この手間を軽減するため，今回の学習活動においては，教師や教材によってサンプルプログラムが提示され，利用者はそれをもとに改造を行っていくスタイルで活動を行った点に着目した．このスタイルの活動においては，利用者の作成するプログラムに似通っている部分が多いと考えられる．そこで，似たプログラム同士を自動分類し，それら似ているプログラムについてはまとめて評価を行えるような評価用のユーザインターフェースを試作した．本発表では，ログに含まれるプログラムを評価した結果と，さらなるログの評価の効率化のための自動分類手法の改善について議論する．

An assistant tool to analyze activity logs in programming lessons for novice students

CHO Shinya[1,a)]　NAGASHIMA Kazuhei[2]　MANABE Hiroki[3]　KANEMUNE Susumu[4]
NAMIKI Mitarou[2]

1. はじめに

筆者らは，プログラミング学習環境として，Web 上で動作可能な BitArrow[1] を開発し，初学者向けの授業におけるプログラミング環境に用いる実践を行っている．

今回の実践では，プログラムの構成要素（変数や制御構造など．以下単に「構成要素」）の働きを理解し，アルゴリズムの実装に役立てられることを目指した．

BitArrow にはログ機能があり，利用者の作成したプログラムや実行結果を，授業担当者からの希望があった場合に限り，ログの形式で Web サーバに保存する機能をもっている．このログを用いて，上記のようなアルゴリズムの実装の活動を通じて，構成要素を活用できたかの評価を行いたい．ここでいう評価とは，利用者のプログラミングの達成度の評価のためというよりかは，実践内容が適切であったかどうか，達成度が思わしくない利用者にどのような支援が必要であるかを明らかにするための「教え方」の評価に主眼を置いている．具体的には，次のような評価の基準（評価項目）が必要であると考えた．

- **「進捗」** 利用者がどのくらい多くのプログラムを書い

[1] 明星大学
Meisei University, Japan
[2] 東京農工大学
Tokyo University of Agriculture and Technology, Japan
[3] 神奈川県立柏陽高等学校
Hakuyo High School, Japan
[4] 大阪電気通信大学
Osaka Electro-Communication University, Japan
[a)] cho@eplang.jp

たか
- **「成否」** プログラムが正しく動作するか
- **「構成要素の使用」** そのプログラム内で構成要素が適切に，過不足なく使用されているか
- **「失敗の原因」** プログラムが正しく動作しない原因（文法的な書き間違い，構成要素の使い方の間違い，考え方・アルゴリズムの間違いなど）
- **「望ましくない書き方」** 正しく動作するが，望ましくないコーディングスタイルや，到達不能なコードなどの冗長な要素があるか

先述したような「教え方」の評価を行なうには，ログの１つ１つ（ログ項目) について，これらの評価を行い，利用者のプログラミングの過程についての詳細な分析が必要であると考えられる．しかし，ログはその量が膨大になり，評価をするのに手間がかかってしまうため，可能な限り自動化を行なうことが望ましい．また，自動化を行なうことによって，BitArrow（学習環境）が自動的に利用者のつまずきを検出し，その場で利用者に適切なアドバイスが行えるような仕組みを構築することにもつながる．

本発表では，自動化の手法を確立する前段階として，筆者らが手動ですべてのログの評価を行い，その中から自動化が可能な評価項目と，手動での評価が必要な評価項目を明らかにする．また，ログを手動で評価した際に，ログを大まかに分類するための手法と，評価をするために作成したユーザインタフェース（以下，評価インタフェース）についても紹介し，これらの仕組みが自動化の支援を行える可能性についても議論する．

2. ログの評価方法

2.1 ログの形式

BitArrow は利用者の作成したプログラムを逐一ログの形式で Web サーバに保存する機能をもっている．ログのデータは，利用者がプログラムファイルを保存したり，プログラムを実行したりしたときに送信される．送信される１つのログ項目に含まれるものは次の通りである．

- 実行または保存を行った日時
- ファイル名
- 保存されたプログラムのソースコード
- コンパイルエラー・実行時エラーの有無．ある場合はエラーメッセージ

2.2 ログへのタグ付け

前述したログからは，保存したファイルの内容の他に，エラーがあった場合のメッセージも含まれている．しかし，エラーがない場合でも，思い通りの結果にならずにつまずいている場合も考えられる．そのような情報はログのデータからは直接知ることはできない．そこで，それぞれのログ項目について，1 章に挙げたような評価項目を「タグ」の形で追加することで評価を行なった．タグの名称の例をいくつか挙げる．

- **「進捗」** そのログ項目において，利用者が書こうとしているプログラムの名称．例えば「プログラム 1」.
- **「成否」** そのログ項目において，プログラムが期待された出力を得る場合「正しい出力」というタグをつける．その上で，プログラム毎に指定された構成要素を適切に使っている場合は「正解」というタグをつける．
- **「構成要素の使用」** そのプログラム内で利用されている構成要素．例えば「条件分岐」「繰り返し」「二重繰り返し」など．ただし，これらは単に使用すればよいものではなく，プログラムの目的に即して適切に利用されている場合のみタグ付けされる．そのため，実際には「繰り返しを使った画像表示」などの具体的な名称になる．
- **「失敗の原因」** 正しくない原因．例えば，「二重ループに同じ変数を使用」,「ループ条件の誤り」（比較演算子の大小が反転しているなど）
- **「望ましくない書き方」** 適切でないコーディングスタイルや，冗長な要素．例えば「for の後ろに中括弧がない」,（`if(x==5){..}else{..}` を `if(x==5){..}else if(x!=5){..}` と書くなどの）「不要な `elseif` 文」

なお，1 つのログ項目に対しては，名称の異なるタグを複数つけることを可能とした．

3. ログの評価支援

今回は，収集したログのログ項目に先述のようなタグを付けることにより，ログの評価を手動で行った．その際に，なるべく手間を減らすための手法をいくつか施した．これらの手法は今後，評価の自動化を行なうにあたっても応用できる可能性が高いためここで詳しく説明する．

3.1 ログ項目の自動分類

プログラミングの初学者は，一からプログラムを作ることは少なく，お手本となるサンプルプログラムを教材などを見て，それを少しずつ改造していく活動スタイルが多い．このことは，利用者の作成するプログラムには似ている点が多いことを意味している．また，その活動を通じて生成されたログのログ項目につけられるタグも似ていると考えられる．

そこで，ログの内容を似ているプログラム同士で自動的に分類を行い，それらの分類されたログ項目毎にタグを付けることで，複数のログ項目をまとめて評価を行えるようにした．

まず，ログ項目に含まれるプログラムのうち，互いに同一と見なされるプログラムは一度しかタグ付けの対象としないことにした．同一であるかどうかは，ログ項目に含まれるプログラムの文字列からコメントやスペースを除去し

た後の文字列がまったく同じであるかどうかで判断した．

同一のログ項目を除去後，次のような手順で，ログ項目を分類した．

- ある ログ項目のプログラムに含まれる字句要素とその出現回数からなる特徴ベクトルを作成する．今回は字句要素として，英数字の並びを抜き出した．
- 各 ログ項目の特徴ベクトルを k-means 法でクラスタリングし，その結果を分類結果とする．今回は $k = 30$ とした．

3.2 タグ付けインタフェース

タグ付けインタフェースの構成を**図 1** に示す．インタフェースは HTML + JavaScript + PHP を用いて Web ブラウザで動作するよう実装した．

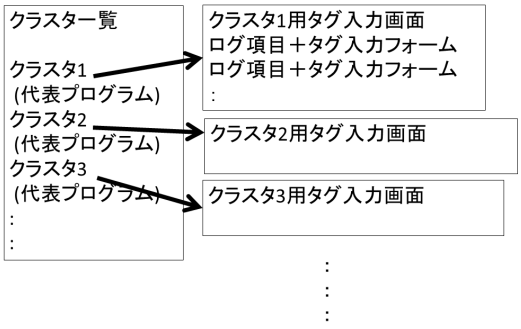

図 1 タグ付けインタフェースの構成

まず，評価者（タグ付けする人）には前述の手法で分類された「クラスター一覧」が提示される．それぞれのクラスタには「代表プログラム」として，そのクラスタの中心ベクトルにもっとも距離が近かったログ項目のプログラムが 1 つ表示される．評価者がクラスタを 1 つ選択すると，**図 2** にあるようなタグ入力画面に遷移する．タグ入力画面は，クラスタごとに独立した画面として生成され，そのクラスタ に属するログ項目を，クラスタの中心ベクトルに近いものから順に並べて表示している．これは，なるべく似ているプログラムを隣接して配置することで，同じタグの組み合わせを連続してつけやすくするためである．

図 2 で「プログラム」と示される領域には，そのログ項目に含まれるプログラムが表示されている．色の付いている部分は 1 つ前のログ項目との差分を示している．

図 2 の「タグ入力フォーム」からタグを入力することができる．タグの内容を入力する欄はテキストボックスとなっており，自由に入力可能であるが，表記ゆれを避けるため，以前に入力したタグについては「タグ候補」から選んで入力をしてもらうようにする．タグ候補は，そこからタグが 1 つ選ばれるたびに，以前に入力されたタグの組み合わせを参照し，テキストボックスにすでに入力されているタグと共起するタグを優先して表示するようにしている．

また，タグ入力欄の左側のボタンは次のような動作をする．

- Check： 入力を完了して次のログ項目に移動
- Same： 1 つ前と同じタグを入力する
- Same&Next： 1 つ前と同じタグを入力し，次のログ項目に移動
- Run: このログ項目に含まれるプログラムを BitArrow 上で実行する

4. 支援機能の評価

4.1 概要

授業担当者から分析の許可を得た上で，BitArrow を利用した高校生 245 名の JavaScript を用いたプログラミング活動のログ収集を行った．

ほとんどの利用者はプログラミングを経験したことがないとため，題材には繰り返しや条件判定など，基本的なアルゴリズムを実装させるプログラムを選んだ．

利用者が実際に書いたプログラムは次のものになる

- 「プログラム 1」: 繰り返しと条件分岐（奇偶判定）を用いて，異なる画像を交互に表示する
- 「プログラム 2」: プログラム 1 を改造し，画像を 10 個表示するごとに改行を行なう

利用者はこの他にも，プログラミングの基本的な仕組みを学ぶためのサンプルプログラムなども入力していたが，プログラム 1，2 以外のプログラムについては，単にその他のプログラムであることを識別するための最低限のタグ付けしか行っていない．

プログラム 1 の正しく動作する例を**図 3**，プログラム 2 の正しく動作する例を**図 4**，**図 5** に示す．

```
for (i=1; i<=50; i++) {
  if (i%2==0) {
    addText("j","<img src=neko1.png>");
  } else {
    addText("j","<img src=bluefish.png>");
  }
}
```

図 3 プログラム 1 の例

4.2 タグ付けの結果

対象の利用者の活動によってログ項目が 28632 個生成された．今回は，これらの中からエラーが起きていないもの（何らかの形で実行ができたもの）22974 個に絞ってタグ付けを行った．

また，3.1 で示したように，プログラムの内容が同一とみなせるものを 1 つのログ項目にまとめたところ，評価す

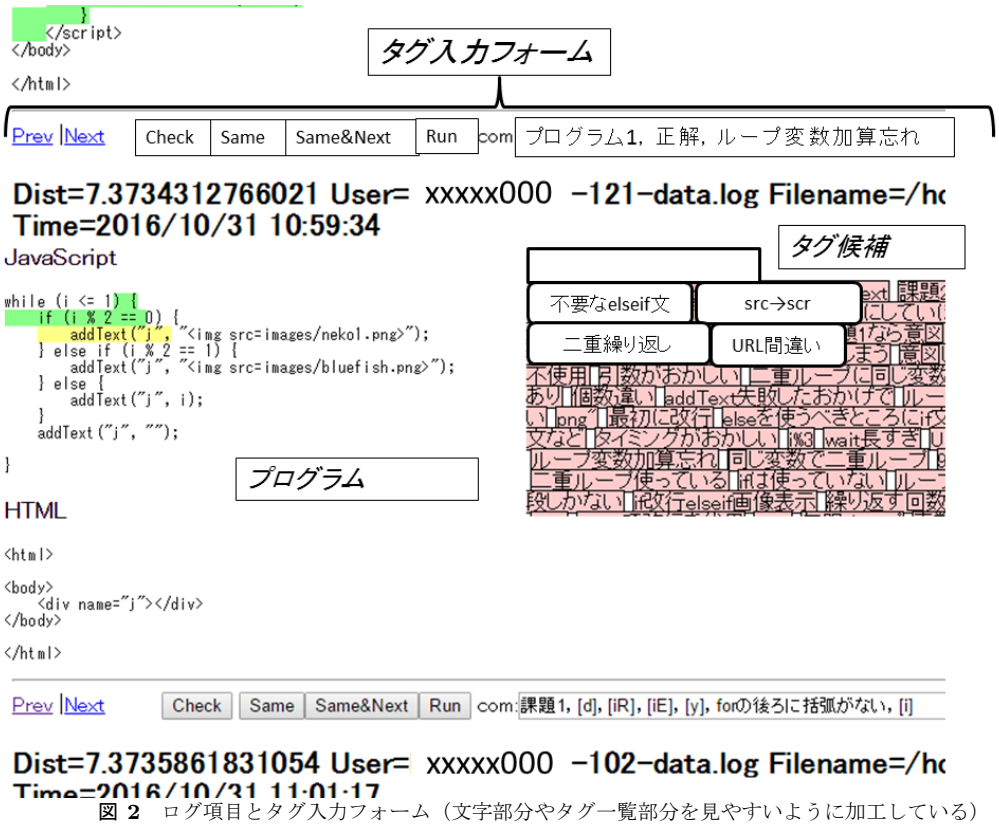

図 2 ログ項目とタグ入力フォーム（文字部分やタグ一覧部分を見やすいように加工している）

```
for (i=1; i<=50; i++) {
  if (i%2==0) {
    addText("j","<img src=neko1.png>");
  } else {
    addText("j","<img src=bluefish.png>");
  }
  if (i%10==0) {
    addText("j","<br>");
  }
}
```

図 4 プログラム 2 の例 (分岐による改行)

```
for (j=1; j<=5; j++) {
  for (i=1; i<=10; i++) {
    if (i%2==0) {
      addText("j","<img src=neko1.png>");
    } else {
      addText("j","<img src=bluefish.png>");
    }
  }
  addText("j","<br>");
}
```

図 5 プログラム 2 の例 (二重繰り返し)

る対象のログ項目の個数は 4889 個となった．さらに，評価するプログラムはプログラム 1，2 のみとしたため，それ以外のサンプルプログラム等とみなせるものの一部にはタグをつけなかったため，評価したログ項目の個数は 3485 個となった．つけたタグの総数は 9487 個であった．

表 1 タグの種類と個数

タグの観点	種類数	総数
「進捗」	5	3207
「成否」	2	1336
「構成要素の使用」	4	2694
「失敗の原因」	126	1269
「望ましくない書き方」	21	981

つけたタグの種類は 158 種類であった．2.2 で述べたタグの観点ごとの種類数を**表 1** に示す．

「進捗」に該当する 5 個のタグのうち，2 つは先述したプログラムの内容の名前を表す「プログラム 1」「プログラム 2」であった．残り 3 つのうち 1 つは「サンプル」であり，これはプログラム 1,2 に入る前の基本的なサンプルプログラムが該当する．残り 2 つは，「サンプルからプログラム 1 に改造中」「プログラム 1 からプログラム 2 に改造中」というものであるが，これらはあるプログラムを作った後，それをもとに別のプログラムに改造している最中であることを表している．

「成否」はプログラムの実行の成否をあらわすもので，2.2 で挙げた「正しい出力」「正解」の 2 つである．

「構成要素の使用」は構成要素の使用の有無であり，プログラム 1 においては，「繰り返しを使った画像表示」と

「分岐による奇偶判定」を，プログラム 2 においては，プログラム 1 のタグに加え「分岐による改行」（図 4 の場合）または「二重繰り返し」（図 5 の場合）をつけた．これらのタグがつけられていることを，「正解」タグがつくための条件とした．

「失敗の原因」は，プログラムの出力が正しくない原因を表す．主なものを**表 2** に示す．

```
for(i=1;i<=50;i++){
    if(i%2==0){
        addText("j","<img src=images/neko1.png>");
        addText("j","<br>");
    } else{
        addText("j","<img src=images/bluefish.png>");
    }
}
```

図 6 「画像表示と改行を同じ if 文で判定」の例

表 3 「進捗」のタグとクラスタ の関係

クラスタ番号	ログ項目数	プログラム 1	プログラム 2
クラスタ 4	302	205	95
クラスタ 12	163	108	48
クラスタ 10	60	11	49
クラスタ 17	483	29	0
クラスタ 8	86	15	0
クラスタ 15	194	11	35
その他の 24 クラスタ	6043	0	0

表 4 プログラム 1 のクラスタ毎の正解数と構成要素

クラスタ番号	項目数	正解	正出力	奇偶判定	繰り返し
クラスタ 4	205	107	127	127	161
クラスタ 12	108	27	91	32	104
クラスタ 10	11	0	9	1	0
クラスタ 17	29	0	3	0	0
クラスタ 8	15	0	0	13	13
クラスタ 15	11	0	11	7	3

5. 考察

ここでは，前節のように手動でつけられたタグを，自動的に付与するための手法，あるいは適切なユーザインタフェースによって手動での付与を支援するための手法を議論する．

5.1 タグの自動検出についての考察

5.1.1 「進捗」の自動検出

「進捗」の自動検出は，あるログ項目について，その利用者が書いているプログラムが何であるかを検出することである．本来は，プログラムの名称などをファイル名などで指定することなどで，ある程度特定ができると考えられるが，今回はファイル名の指定は行っておらず，また指定をしたとしても，間違って名前をつけたり，途中で名前を変更したりする可能性を考えて，今回はプログラムの内容のみで「進捗」を検出することを考える．プログラムの内容は，その中で使われる要素や関数などが異なってくることから，使用される字句要素の違いがプログラムごとに特徴づけられると考えた．したがって，3.1 の自動分類の結果を用いて「進捗」を自動検出できるかを考える．

すべてのログ項目のうちいくつか（9236 個）について，自動分類の結果（どのクラスタ に分類されたか）と，実際に付けられた「進捗」のタグとを比較するため，「プログラム 1」「プログラム 2」のいずれかのタグがついたログ項目と，分類されたクラスタ との関係を**表 3** に示す．

クラスタの個数は 30 であるが，「プログラム 1」「プログラム 2」がつけられたクラスタは 6 個だけであり，クラスタリングにより，サンプルプログラムなどのそれ以外のプログラムとは区別がなされていることがわかる．ただし，これら 6 個のクラスタの中で，表 3 下部のクラスタ 17，8，15 など一部のクラスタにおいては，プログラム 1，2 に取り組む際に，サンプルプログラムを改造しながら作っていたために，プログラム 1，2 に着手しているとはいいきれないものも含まれていた．

今回の自動分類においては，字句要素を用いたクラスタリングという，極めて単純な手法を用いているものの，この手法が利用者が着手しているプログラムの種類，ひいては進捗を知るための手がかりとなる可能性が示唆された．

5.1.2 「成否」および 「構成要素の使用」の自動検出

前述のプログラム 1，プログラム 2 に分類された各クラスタについて，「成否」に属する「正しい出力」および「正解」タグ，および「構成要素の使用」に属する「繰り返しを使った画像表示」「分岐による奇偶判定」「分岐による改行」「二重繰り返し」がつけられた個数を**表 4**，**表 5** に示す．

表 4 で「正解」となる条件は，「正しい出力」「繰り返しを使った画像表示」「分岐による奇偶判定」がすべてつけられていること，表 5 で「正解」となる条件は，「正しい出力」「繰り返しを使った画像表示」「分岐による奇偶判定」がすべてつけられており，さらに「分岐による改行」または「二重繰り返し」がつけられていることとである．

プログラム 1 は「正解」のログ項目を含むクラスタはクラスタ 4，12 の 2 つしかないことがわかる．それ以外のクラスタにおいては，正しい出力ができているものも含まれているが，「繰り返しを使った画像表示」や「分岐による奇偶判定」が書けていないため不正解になっているものが分類されている．

このことから，今回適用した自動分類（クラスタリング）の手法について，次のことが言える．

- クラスタリングによって「正解の多いクラスタ」「不

表 2 「失敗の原因」の例

名称	個数	説明
必ず改行してしまう	211	プログラム 2 において，画像 1 個を表示するたびに改行する
二重ループに同じ変数使用	73	二重繰り返しにおいて，外側と内側に同じ変数を使用している
for の後ろに括弧がない	70	for 文の繰り返し部分を {} で囲んでいないため，繰り返しの範囲が意図したものと違う
画像表示と改行を同じ if 文で判定	68	プログラム 2 において，**図 6** のように改行のための if 文と，画像表示のための if 文が同じになっている
ループ条件の誤り	55	繰り返しにおける条件を間違えている．例えば不等号の向きを逆にしている，あるいは，繰り返しの「継続条件」を書くべきところに「終了条件」を書いている．
ループ変数加算忘れ	50	while 文の中で加算処理 (`i++;` など) がない
URL 間違い	49	`<img>`タグで画像を表示するときに URL を間違えている
> がない	41	`<img>`タグで画像を表示するときに最後の閉じ括弧がない
src → scr	38	`<img>`タグで画像を表示するときに属性名を間違えている
個数違い	29	プログラム 2 において，改行を判定するときの条件が違う．`i%10` を `i%20` と書くなど．
++→ tt	20	`++`演算子を `tt` と書いている
判定する変数と更新する変数が違う	19	プログラム 1 で，ループ変数が `i` であるが，判定部分で `if (n%2==0)` と書いているなど
メソッド名間違い	12	メソッド名を間違えている
初期化なし	10	while 文においてループ変数を初期化していない

表 5 プログラム 2 のクラスタ毎の正解数と構成要素

クラスタ番号	項目数	正解	正出力	奇偶判定	繰返画像	分岐改行	二重繰返
クラスタ 4	95	6	6	72	89	8	6
クラスタ 12	48	0	15	2	0	46	22
クラスタ 10	49	0	43	29	0	0	0
クラスタ 15	35	0	28	15	0	0	0

正解だけを集めたクラスタ」を抽出することは可能

- 「正解の多いクラスタ」にも不正解のものも含まれているため，「正解だけを集めたクラスタ」を作成することは困難

「正解だけを集めたクラスタ」を作成するには，正解の多いクラスタから，不正解のログ項目をクラスタリング以外の手法と組み合わせて検出・排除していく必要があると考えられる．

まず，正解かどうかの必要条件である「正しい出力」を自動的に判定する手法については，プログラミング教育において，学習者の書いたプログラムを自動採点する仕組みがすでに提案されている [2]．このような仕組みでは，学習者の書いたプログラムの出力結果と，模範解答の出力結果と比較して同じ出力が得られるかで判定を行なっている．ただし，今回の実践では，HTML を用いて画像を出力するプログラムであったため，見かけ上同じ出力でも異なる書き方が多様にできてしまうため，単純に模範解答と比較することができないという問題がある．

今後は，次のような方法が必要になると考えられる．

- 出力結果の HTML に対して，正規表現などのパターンマッチを行い，明らかに出力が正しくないものを排除する
- 残った HTML をブラウザ等で一覧し，実行結果を目視で検査する．このとき，なるべく採点者に余分な操作をさせないように UI を工夫する必要がある

また，今回は，出力が正しいかだけではなく，プログラムごとに指定された構成要素を活用しているかどうかも正解の条件となっている．学習者の書いたプログラムにおける構成要素の使用の有無を自動判定する仕組みも提案されている [3] が，単に要素を使用しただけではなく，プログラムの目的に沿って使用されているかどうかは機械な判定が難しい．そのため，プログラムの種類ごとに検出方法を変える必要がある．これについては次節でも触れる．

5.1.3 「失敗の原因」の自動検出

「失敗の原因」のタグの検出，利用者のつまずきの原因を特定するうえで重要であるが，その原因が多岐にわたるため，決まった方法で検出することが難しいと考えられる．また，プログラムの種類によって検出方法を変える必要があるものと，多くのプログラムにに共通して同じ方法で検出できるものもある．

そこで，表 2 で挙げたタグについて，検出方法と汎用性の観点で分類したものを**表 6** に示す．

「検出方法」は次のいずれかである．

- 「行レベル」: 対象プログラムのある一行ついて調べれば高確率で自動検出できるタグ（検出が比較的簡単）
- 「構文レベル」: 対象プログラムの複数の行，あるいは構文（制御構造の入れ子関係など）をみて検出する必要があるタグ（検出が比較的困難）

「汎用性」は次のいずれかである．

- 「高」: 多くのプログラムに共通して，同じ方法で検出

表 6 「失敗の原因」の検出方法の分類

名称	検出方法	汎用性
必ず改行してしまう	構文レベル	プログラム依存
二重ループに同じ変数使用	構文レベル	高
for の後ろに括弧がない	構文レベル	高
画像表示と改行を同じ if 文で判定	構文レベル	プログラム依存
ループ条件の誤り	行レベル	プログラム依存
ループ変数加算忘れ	構文レベル	高
URL 間違い	行レベル	環境依存
> がない	行レベル	プログラム依存
src → scr	行レベル	環境依存
個数違い	行レベル	プログラム依存
++→ tt	行レベル	高
判定する変数と更新する変数が違う	構文レベル	プログラム依存
メソッド名間違い	行レベル	環境依存
初期化なし	構文レベル	高

可能なタグ

- 「環境依存」: 特定の環境（HTML の出力），特定の API(`addText` など）を用いたプログラムであれば同じ方法で検出できるタグ
- 「プログラム依存」: 特定の種類のプログラムにしかつけられないタグ，あるいはプログラムによって検出方法を変えなければならないタグ

これらの分類をもとに，タグ付けの自動化に向けて次のような方針を立てることにした．

- 汎用性「高」または「環境依存」のタグは，将来に渡って（他のプログラムでも）同じような間違いをする利用者が見込めるため，検出方法の難易度にかかわらず，BitArrow（学習環境）に組み込んでおく．
- 汎用性「プログラム依存」のタグは，実習をして初めてその内容（検出方法）が明らかになるものであるため，教える側が学習環境に対して動的に検出方法を追加できるような仕組み [4] を作っておくことで，他の学習者と同じ点でつまずいている学習者が現れたときに，自動的にアドバイスできるようになると期待される．ただし検出方法については検討が必要である：
  - 検出方法が「行レベル」であれば，簡単な検索式（文字列または正規表現等）でその検出方法を素早く追加できると考えられる．
  - 検出方法が「構文レベル」のものは，文字列や正規表現等で検出するのは困難であるため，プログラムの構文を考慮した検索が簡便に行えるような仕組みを新たに開発する．これは，以前に開発した，プログラミング教材の構文をチェックするツール [5] を改良して対応する予定である．

なお，5.1.2 で挙げた「構成要素の使用」に属するタグについても，上記のような方法で自動検出することで，正解となるプログラムを自動的に検出することが期待できる．

表 7 タグ修正距離と該当したログ項目数（クラスタ 4,12,10）

タグ修正距離	項目数
0	82
1	33
2	27
3	26
4	37
5	29
6	33
7	14
8 以上	11

5.2 評価インタフェースの考察

今回，手動によるタグ付けのために作成した評価インタフェースは，類似するログ項目をなるべく近くに集めて，タグ付けをまとめて行える仕組みを取り入れた．図 2 において，各 ログ項目のタグ付け欄には「Same」というボタンがあり，これを押すことで，1 つ前にタグ付けしたログ項目と同じタグを入力することができ，修正が必要な部分だけを修正することができる．さらに，1 つ前とまったく同じタグの組み合わせをつけたい場合，「Same&Next」ボタンを押して，修正することなく次のログ項目のタグ付けにすぐ移動するようにしてある．これらのボタンを多用することでタグ付けが効率化されるようになっている．

この機能が働くためには，同じ組み合わせのタグがつくログ項目が評価インタフェース上で隣接して配置される必要がある．ここでは，タグ付けされた結果から，隣接するログ項目同士の「タグ修正距離」を集計した．タグ修正距離とは，あるタグの組み合わせを別のタグの組み合わせに変更する際に必要な「削除するタグの個数」と「追加するタグの個数」の合計とする．タグ修正距離が 0 の場合，直前のログ項目と同じ組み合わせのタグをつけたことになり，「Same&Next」ボタンを押すことが可能であったと考えられる．また，タグ編集距離が 0 でない場合も，その距離が小さいほど，「Same」ボタンを押した際に修正する手間が省けたと考えられる．

プログラム 1,2 を評価した個数の多かったクラスタ 4,12,10 の評価インタフェースにおいて，タグ修正距離の分布を**表 7** に示す．なお，当該評価インタフェースにおいてタグ付けしたログ項目の個数は 292[*1]，ログ項目 1 個あたりにつけられたタグの平均個数は 4.1 であった．

この結果から，タグ修正距離が 0，すなわち「Same&Next」ボタンが押せたであろう回数は 82 回（全体の 28%）であり，また，半数近くのログ項目において，タグ修正距離が 2 以内であったため，「Same」ボタンも高頻度で活用されたことが言える．

しかし，実際の評価インタフェースにおけるタグの並び

[*1] プログラムが同一のログ項目には 1 回しかタグづけしないため，表 3 の個数より少なくなる

を見ると，同じタグの組み合わせをもつログ項目の間に，別の組み合わせのタグがつけられた ログ項目が割り込んでいたりすることが多く見られた．特に多かったのが，「for の後ろに括弧がない」が入っている以外は同じ組み合わせのタグが連続している並びであった．これは，クラスタリングをしたときに，特徴ベクトルを作る際に対象にしていた字句要素が英数字の並びだけであったため，中括弧記号の有無を無視してクラスタリングしたためと考えられる．

5.3 クラスタリング手法の考察

今回，ログを分類するための手法として，「字句要素の抽出」「ベクトル化」「k-means 法によるクラスタリング」を実施したが，あくまでこの手法は手動によるタグ付けのための暫定的なものであり，今後以下のような観点でより詳細に手法を検討する必要がある．

- **字句要素をどのように選ぶか**
  今回，字句要素として選んだのは，識別子のみであった．これは，JavaScript や HTML などの複数の言語において，字句要素を取り出すのに簡便であったためであるが，先述のように括弧の要素の有無を見落として，本質的に意味の異なるプログラムを同一視してしまうなどの問題点があった．また，プログラミングにおいては，制御構造が入れ子になっている場合に同じ文でも異なる働きをすることも考えられる．今後は，字句要素として含めるものを増やしたり，同一の字句要素を制御構造の階層（中括弧の深さなど）によって別のものとして扱うなどの手法を取り入れる予定である
- **ベクトル化を何に拠って行なうか**
  今回は，字句要素の出現頻度をそのままベクトルにしてクラスタリングを行ったが，字句要素の重みを考慮する（例えば，tf-idf 法など）ほうがよりよいクラスタリング結果になる可能性も考えられるため，重み付けの手法を検討していく．
- **クラスタリング手法およびそのパラメタ**
  今回のクラスタリングにおいては，比較的高速な k-means 法を用いた．$k = 30$ としたが，この値が適切かどうかは検討を要する．また，1 つのクラスタの中に異なるタグ付けの結果をもつログ項目が見られたことから，1 つのクラスタをさらに階層的にクラスタリングすることで，特定のタグが集まりやすいクラスタを見つけられる可能性もある．

5.4 BitArrow への今後の機能拡張

今後，タグ付けの自動化の手法を確立すると同時に，自動化した結果を利用者に次のような形でフィードバックし，利用者のプログラミング上のつまずきを解決する機能を拡張していく

- 活動に伴って蓄積されるログ項目をリアルタイムにクラスタリングし，活動が進行するにつれて利用者の書いたプログラムの傾向を俯瞰する
- クラスタリングした結果から クラスタの代表を見せて，教える側が即時対応できるようにする
- 教える側がタグの判断基準を動的に追加することで，それ以降に生成されたログ項目にはタグが自動的に追加されるようになる
- ログにタグが付与されるとともに，該当利用者にタグをもとにしたアドバイスを学習者の画面に提示する

6. まとめ

Web ベースのプログラミング学習システム BitArrow のログ収集機能を利用し，利用者のプログラミング活動のログを記録した．今後のプログラミング学習活動の改善のため，特に利用者のつまずきの原因を探るため，ログの内容を手動で評価（タグ付け）した．その際，評価の手間を減らすための大まかな分類を自動的に行い，評価の支援を行なうユーザインタフェースを作成した．

本発表においては，実際に付けられたタグの内容から，これらのタグをログに自動付与するための手法を考察し，今後の BitArrow への機能拡張に向けた議論を行った．

参考文献

[1] 長島和平，本多佑希，長　慎也，間辺広樹，兼宗　進，並木美太郎：オンラインで複数言語を扱うことができるプログラミング授業支援環境，情報教育シンポジウム 2016 論文集，Vol. 2016, pp. 1–9 (2016).
[2] 太田翔也，富永浩之：プログラミング演習における補助者の巡回指導のためのタブレット PC 上の支援ツール - 小コンテスト形式の初級 C 演習での実践におけるツールの操作ログの分析 - ，情報処理学会研究報告，Vol. 2016-CE-137, No. 1, pp. 1–8 (2016).
[3] 太田　剛，森本容介，加藤　浩：プログラム機能の自動分析機能とプログラム概念の自動評価機能を持つ Scratch 用プログラミング学習支援システムの開発，情報教育シンポジウム 2016 論文集，Vol. 2016, pp. 106–113 (2016).
[4] 長　慎也，筧　捷彦：proGrep ー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コンピュータと教育（CE），Vol. 2005, No. 15, pp. 29–36（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110002777102/⟩ (2005).
[5] 長　慎也，保福やよい，西田知博，兼宗　進：De-gapper—プログラミング初学者の段階的な理解を支援するツール，情報処理学会論文誌，Vol. 55, No. 1, pp. 45–56 (2014).