一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
ET2020-17(2020-09)

プログラミング学習時の学習履歴を活用した論理エラーの分析

梅澤　克之† 　中澤　　真†† 　小林　　学††† 　石井　雄隆†††† 中野美知子†††
平澤　茂一†††

† 湘南工科大学　〒 251–8511 神奈川県藤沢市辻堂西海岸 1–1–25
†† 会津大学短期大学部　〒 965–0003 福島県会津若松市一箕町大字八幡門田 1–1
††† 早稲田大学　〒 169–8555 東京都新宿区大久保 3–4–1
†††† 千葉大学　〒 263–8522 千葉県千葉市稲毛区弥生町 1-33
E-mail: †umezawa@info.shonan-it.ac.jp, ††nakazawa@jc.u-aizu.ac.jp, †††{mkoba,nakanom,hira}@waseda.jp,
††††yishii@chiba-u.jp

あらまし　学習ログを蓄積することで学習者がプログラムのどこをどのように修正したかを確認できる編集履歴可視化システムを提案してきた．このシステムを反転授業に活用し，膨大な学習ログを蓄積した．この学習ログは，プログラムが完成するまでに修正されていく過程の全てのソースコードを含んでいる．この学習ログを参照し文法エラーを含むデバッグ練習用の問題を自動生成するデバッグ練習問題抽出システムを開発した．本報告では，大量のプログラミング学習履歴を蓄積し分析することによって間違いを起こしやすい論理エラーを自動で検出することを目指す．
キーワード　プログラミング，編集履歴，学習ログ，デバッグ練習

Analysis of logic errors using learning history during programming learning

Katsuyuki UMEZAWA†,
Makoto NAKAZAWA††, Manabu KOBAYASHI†††, Yutaka ISHII††††, Michiko NAKANO†††, and
Shigeichi HIRASAWA†††

† Shonan Institute of Technology　Tsujido-Nishikaigan 1–1–25, Fujisawa, Kanagawa, 251–8511, Japan
†† Junior College of Aizu　Monden 1-1, Yahata, Ikki-Machi, Aizuwakamatsu, Fukushima 965–0003, Japan
††† Waseda University　Okubo 3–4–1, Shinjuku, Tokyo 169–8555, Japan
†††† Chiba University　Yayoi-cho 1-33, Inage-ku, Chiba-shi, Chiba, 263–8522 Japan
E-mail: †umezawa@info.shonan-it.ac.jp, ††nakazawa@jc.u-aizu.ac.jp, †††{mkoba,nakanom,hira}@waseda.jp,
††††yishii@chiba-u.jp

Abstract　We have proposed an editing record visualization system which can confirm where and how the learner modified program by storing learning log. We utilized this system for actual flipped classroom and stored enormous learning logs. This learning log contains all the source code in the process of being modified until the program is completed. We developed a debugging exercise extraction system to automatically generate problems with syntax errors for debugging practice from this learning log. In this report, we propose a method to identify the logic error in which error information is not output by analyzing the learning log.
Key words　Programming, Editing Record, Learning Log, Debugging Practice

1. ま え が き

近年，学習モデルは多様化し，e ラーニングから，教場授業と e ラーニングを併用するブレンディッドラーニングまで多岐にわたっている．このような教育環境においては，LMS（学習管理システム）が利用される．LMS は学習コンテンツの配信，試験の実施と採点など様々な機能を持ち，また LMS を経由して学習者の学習活動に関する多くの履歴（操作ログなど）・記録（レポート・評価結果など）の収集が可能である．

我々は，プログラミング言語学習の学習環境である編集履歴

可視化システムを提案してきた [1]．このシステムは学習環境の準備を容易にするとともに学習者の状況把握することができる．また学習ログを蓄積することで学習者がプログラムのどこをどのように修正したかを確認できる．また我々は，自習時の学習ログに基づき対面授業時にグループ分けをして授業を行うグループ分け反転授業を提案してきた [2] [3]．この反転授業を実授業に適用するにあたり，前述の編集履歴可視化システムを活用した [4] [5]．その結果，3 年間で約 600 名分の膨大な学習ログが蓄積された．編集履歴可視化システムでは，プログラムが完成するまでに修正されていく過程のソースコードをすべてログとして蓄積している．これらの情報を元にあえて文法エラーを含むソースコードを学習者に与えて，そこに含まれている間違いを修正するデバッグ練習用の問題を自動生成するシステムの開発を行った [6] [7]．

これに対して本報告では，同じ学習履歴を利用して，論理エラーの分析を行う．論理エラーは文法的には誤りはないためコンパイラはエラー情報を出力しない．よって機械的に検出することは難しい．文献 [8] では，学習ログを観察することでいくつかの論理エラーを数え上げている．しかし膨大な学習ログを対象とする場合には，人手による観察では対応できない．本報告では，大量のプログラミング学習履歴を蓄積し分析することによって間違いを起こしやすい論理エラーを自動で検出することを目指す．

2 章では，関連技術として，編集履歴可視化システムの概要について述べる．3 章で既存のデバッグ練習問題抽出システムについて，全体構成，および抽出アルゴリズムについて記述する．4 章で論理エラーを検出するための方法について示し，5 章で実験の方法を分析結果について示す．6 章でまとめと今後の課題を示す．

2. 関 連 技 術

2.1 学習履歴の蓄積システム

学習者がプログラムを完成させて行く過程や，その過程でどのような課題に直面し，どのように解決したかを知ることは難しく，それらを理解するためにはバージョン単位の履歴では不十分であり，どのように変更されてきたのかユーザの操作単位で詳細に記録を取ることの有効性が指摘されている [9] [10]．文献 [9] では，蓄積した学習履歴の活用として，操作履歴を表示する機能，表示機能に対して表示条件を設定してフィルタリングする機能，過去の任意の状態のソースコードを表示・復元する機能を提案している．文献 [10] では，詳細な学習履歴をリアルタイムに蓄積し分析することで，該当の学習者への良い点，悪い点のフィードバックを可能にし，教師側は学習者の躓きの箇所がわかる機能を有しているが，蓄積した学習者全員の履歴を自動で分析したり，分析結果をもとに品質を向上させるまでには至っていない．

2.2 編集履歴可視化システム

我々も 2.1 節に示したシステムと同様のコンセプトに基づいて，プログラミング言語学習の学習環境である編集履歴可視化システムを提案してきた [1]．

編集履歴可視化システムは，プログラミング言語学習（英語学習にも適用可能）の初学者に特化した学習環境である．このシステムは，学習環境の準備の容易化，学習者の状況の把握などの課題を解決するためのシステムである．「学習環境の準備の容易化」に関しては，開発環境のインストールなどの前準備は不要で，ブラウザから利用可能となっている．そのため学習者は，PC だけでなくスマートフォンやタブレットからの学習も可能である．また「学習者の状況把握」に関しては教師用の画面でソースコードや実行結果の 1 つ前の状態に対する差分を表示可能である．この差分を見れば学習者がどこをどう修正したかを確認できるようになっている．また，それぞれの学習者が何問目の問題を解いている最中なのかを一目で確認できるようになっている．開発可能なプログラミング言語は，C/C++ [11], Java, HTML/JavaScrip, Scratch [12] などである．また英語のライティング問題にも対応している [13]．Java 言語版の編集履歴可視化システムの学習用画面を図 1 に，教師用の確認画面を図 2 に示す．

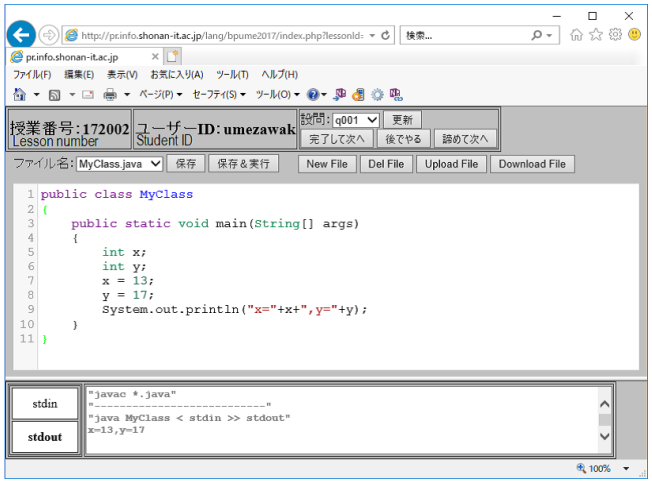

図 1　編集履歴可視化システム（学習用画面）

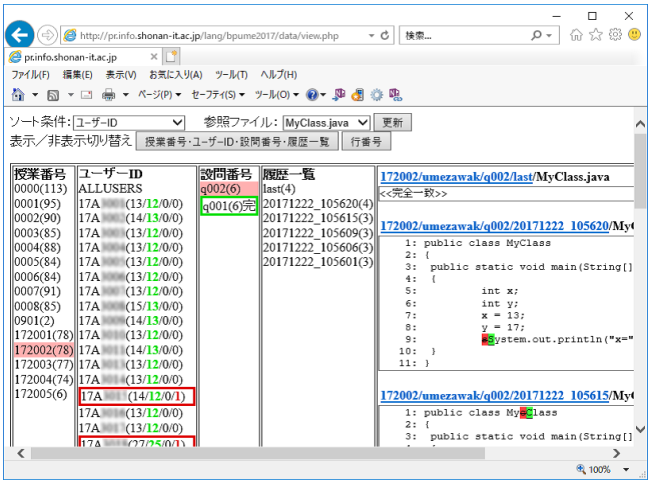

図 2　編集履歴可視化システム（教師用画面）

3. デバッグ練習問題抽出システム

我々は文法エラーを自動的に抽出するデバッグ練習問題抽出

システムを開発した [6] [7]．本節では文法エラーを対象としたデバッグ練習問題抽出システムについて簡単に記載する．

3.1 システムの全体構成

文法エラーを対象としたデバッグ練習問題抽出システムの全体構成を図 3 に示す．図 3 に示す通り，学習履歴を蓄積するまでは既存システムである編集履歴可視化システムを活用する．デバッグ練習問題抽出ツールは，編集履歴可視化ツールが蓄積した学習履歴を参照し，正解のソースコードと正解に至るまでの誤りを含むソースコードを比較し，誤りを含むソースコードを抽出する．抽出された誤りを含むソースコードは，編集履歴可視化システムの改造版で参照及び配布される．

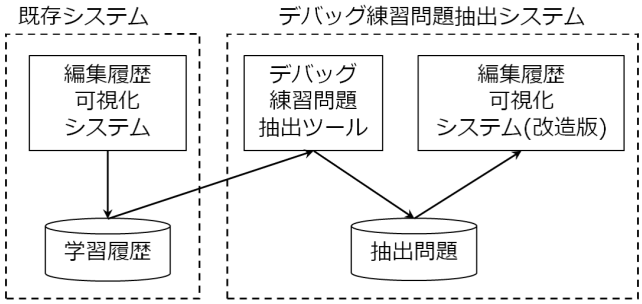

図 3　デバッグ練習問題抽出システムの全体構成

3.2 デバッグ練習問題抽出ツール

デバッグ練習問題抽出ツールは，編集履歴可視化ツールが蓄積した学習履歴，具体的には，各問題ごとの完成フォルダ（last）に存在する MyClass.java と，作成途中の MyClass.java を比較し，修正箇所数および各修正箇所の修正文字数を抽出する．抽出した「修正箇所数」および「最大修正文字数」をもとにフォルダを再構成する．なお，ソースコードの差分には，エラーを修正したための差分と，エラーを伴わない差分があるため，エラーを修正した場合のソースコードのみ抽出する事としている．

3.3 抽出アルゴリズム

本節では抽出アルゴリズムを示す．

1) 図 4 のフォルダ構成の全履歴について下記を繰り返す．
2) last.info に “end” と記載されているかを確認する．
3) last フォルダ以外のフォルダについて，stdout ファイルに “エラー” が記載されているかどうかを確認する．
4) last フォルダ以外のエラーが記載されているフォルダの MyClass.java と last フォルダの MyClass.java を差分をとる(注1)．
5) その際に，差分の箇所の数と 1 つの差分に最大何文字含まれているかをカウントする．
6) 差分の箇所数と 1 つの差分に最大何文字含まれているかによって図 5 のフォルダ構成に従って MyClass.java ファイルをコピーする．

(注1)：Java 版編集履歴可視化システムでは，main 関数を含むクラス名は MyClass でありそのファイル名は MyClass.java と規定されている．

```
+ 0001（レッスン番号）
+ 0002（レッスン番号）
    + 17Axxx1(学籍番号)
    + 17Axxx2(学籍番号)
        + q001(問題番号)
        + q002(問題番号)
            + 20171006_092309(履歴)
            + 20171006_092508(履歴)
                - MyClass.class
                - MyClass.java
                - stdin
                - stdout
            + last (最終履歴)
                - MyClass.class
                - MyClass.java
                - stdin
                - stdout
            - last.info(完了ステータス)
```

図 4　編集履歴可視化システムのフォルダ構成

```
+ 0001（レッスン番号）
+ 0002 （レッスン番号）
    + q001(問題番号)
    + q002(問題番号)
        + 01-01-000001(個所数-文字数-連番)
        + 01-01-000002(個所数-文字数-連番)
        + 01-01-000003(個所数-文字数-連番)
            + before
                - MyClass.java
            + last
                - MyClass.java
        + 02-01-000004(個所数-文字数-連番)
            + before
                - MyClass.java
            + last
                - MyClass.java
```

図 5　デバッグ練習問題抽出ツールが出力するフォルダ構成

4. 論理エラー対応

4.1 概　　要

3.3 節で示したアルゴリズムの中でプログラム学習履歴の中に “エラー” が記載されているかどうかを確認するステップがある．これを確認することにより文法エラーを起こしているプログラムソースファイルを特定しようというアルゴリズムである．これに対して今回実現したいことは論理エラーを分析したいことである．

前述のアルゴリズムでは文法エラーを対象にしたため学習履歴に “エラー” が記載されているプログラムソースファイルの集合を対象とした．逆に学習履歴に “エラー” が記載されていないプログラムソースファイルの集合には，論理エラーを含むプログラムソースファイルが含まれている（論理エラーを含まないプログラムソースファイルも含まれる）．それらを分析することで学生が誤りを起こしやすい論理エラーを特定することが出来ると考えた．

4.2 出力結果の考察

既存のデバッグ練習問題抽出ツールのアルゴリズムを変更し，学習履歴に “エラー” が記載されていないソースファイル

と最終ソースファイルとの差分を出力させた．出力結果の例を図 6,7,8 に示す．図 6 はダブルクォーテーションに挟まれた文字列の修正である．文法エラーとはならないが課題によっては表示方法が正当とはならず論理エラー扱いになる場合がある．図 7 は if 文の条件部の修正であり，図 8 は，代入文の修正である．こちらも文法エラーではなく論理エラーの部類である．

```
1: public class MyClass
2: {
3:     public static void main(String[] args)
4:     {
5:         System.out.println("Hello wWorld");
6:     }
7: }
```

図 6　文字列の変更

```
1: import java.io.*;
2:
3: public class MyClass
4: {
5:     public static void main( String[] args ) throws IOException
6:     {
7:         int x,y;
8:         x = 5;
9:         y = 3;
10:        if(x>=y){
11:            System.out.println( + x +"は" + y + "より大きい");
12:        }
13:    }
14: }
```

図 7　if 文の条件部の修正

```
1: import java.io.*;
2:
3: public class MyClass{
4:     public static void main(String[] args) throws IOException{
5:         int x, y ,z;
6:
7:         x=3;
8:         y=7;
9:
10:        z=xy;
11:        x=yy=x;
12:        y=x=z;
13:        System.out.println("x="+ x +", y="+ y);
14:    }
15: }
```

図 8　代入文の修正

4.3　論理エラーの種類

前節で示したように，出力された差分情報を見るといくつかの種類の論理エラーが含まれていることが分かった．前節の 3 種類以外にも表 1 に示すような論理エラーを含む修正が行われていることが分かった．

4.4　論理エラーに対応したデバッグ練習問題抽出ツール

前述の分析では目視によりソースファイルの差分情報を確認したが，既存のデバッグ練習問題抽出ツールに対して自動で論理エラーの種別を分類する機能を追加した．開発したツールを図 9 に示す．

5.　実験と分析

5.1　データ収集を行った授業について

我々は，反転授業の自習時の e-ラーニングの学習ログを取得し，自習時の理解度が高い学生のグループ，自習に時間をかけなかったために理解度が低い学生のグループ，自習に時間をかけたが理解度が低い学生のグループに分けて教場での対面授業を行う「グループ分け反転授業」を提案してきた [2] [3]．このグループ分け反転授業の方式を 2017 年度後期における 16 週間の

表 1　論理エラーの種類

種類	説明
空白	スペースやタブの追加や削除
コメント	//によるコメントの追加や削除
文字列	"と"に囲まれた文字列の変更
括弧	(,),{,}の追加や削除
for 文	for 文そのものの書き換えと条件部の修正
while 文	while 文そのものの書き換えと条件部の修正
if 文	if 文そのものの書き換えと条件部の修正
else 文	else 文そのものの書き換え
println	println,printf,print の書き換えと () の中身の修正
セミコロン	; の追加と削除
配列	配列のサイズと添え字の変更
変数	変数の変更
数字	数字の変更
代入文	代入文の変更
式	式の変更
その他	上記に分類できない修正

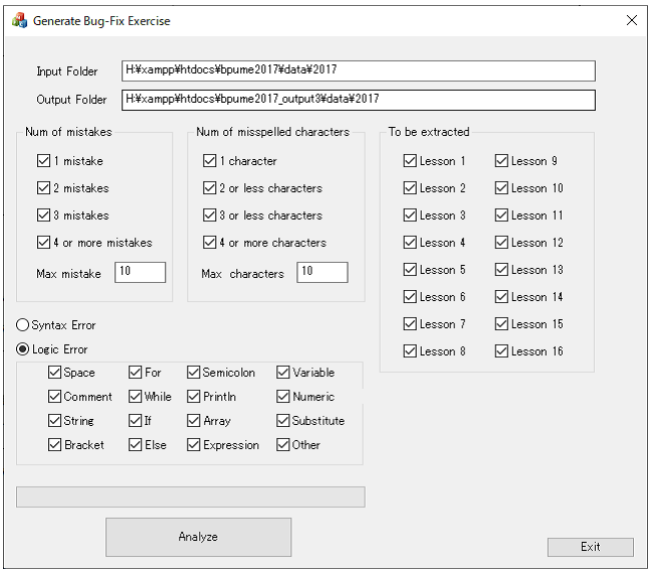

図 9　論理エラーに対応したデバッグ練習問題抽出ツール

実授業「基礎プログラミング実習」に適用し，自習後の理解度と対面授業後の理解度の観点でその有効性を示してきた [4] [5]．この反転授業の実授業への適用の中で，2.2 節で示した Java 言語用編集履歴可視化システムを活用し，2017 年度，2018 年度，2019 年度の 3 年間で膨大な学習ログを蓄積した．なお，「基礎プログラミング実習」で行った Java 言語の授業の内容と学習者に課した課題数を表 2 に示す．

5.2　分 析 結 果

前節で示した 8 週間の授業において各週には表 2 に示した数の課題を課した．これらの課題を解くプログラムを作成する過程で「ビルド＆実行」ボタンを押すごとにその時のソースプログラムがログとして蓄積される．今回の分析は，蓄積された 203,436 個（2017 年の 72,193 個，2018 年の 61,721 個，2019 年の 69,522 個）のソースファイルを対象に分析を実施した．

各課題を解く際に，一つ前の問題のソースコードをコピーし

表 2　授業内容の全体説明
Table 2　Description of class

	内容	課題数
1 週目	Java 言語（入出力）	7
2 週目	Java 言語（変数・演算）	12
3 週目	Java 言語（分岐）	13
4 週目	Java 言語（繰り返し）	22
5 週目	Java 言語（配列）	8
6 週目	Java 言語（メソッド）	8
7 週目	Java 言語（クラス I）	6
8 週目	Java 言語（クラス II）	4

てそこから修正する学生が多い．その場合には，コピーしたソースコードから大幅に修正を加えてから [ビルド＆実行] を行うということが想定される．つまり完成版のソースコードと比較してあまりにも多くの箇所数の修正，あるいはあまりにも長い文字数の修正は，論理エラーとしての対象とするのはふさわしくない．よって「修正箇所数」および「最大修正文字数」が 10 より大きいソースコードは論理エラーとしての検出対象外とした．

表 3 に論理エラー種別毎の検出数と年度の合計に対する割合 (%) を示す．

表 3　論理エラー種別ごとの検出数

種類	検出数 (割合 %)					
	2017		2018		2019	
空白	4,189	(19.94)	2,735	(15.12)	2,934	(14.85)
コメント	124	(0.59)	34	(0.19)	289	(1.46)
文字列	2,616	(12.45)	2,675	(14.78)	2,410	(12.20)
括弧	1,140	(5.43)	1,164	(6.43)	1,228	(6.21)
for 文	2,771	(13.19)	2,223	(12.29)	2,577	(13.04)
while 文	152	(0.72)	108	(0.60)	152	(0.77)
if 文	1,453	(6.92)	1,122	(6.20)	1,384	(7.00)
else 文	49	(0.23)	31	(0.17)	183	(0.93)
Println	89	(0.42)	101	(0.56)	93	(0.47)
セミコロン	869	(4.14)	649	(3.59)	649	(3.28)
配列	371	(1.77)	333	(1.84)	454	(2.30)
変数	3,215	(15.30)	2,476	(13.68)	2,390	(12.10)
数字	1,447	(6.889)	2,006	(11.09)	2,313	(11.71)
代入文	279	(1.33)	252	(1.39)	331	(1.68)
式	2,220	(10.57)	2,155	(11.91)	2,333	(11.81)
その他	26	(0.12)	29	(0.16)	40	(0.20)
合計	21,010	(100)	18,093	(100)	19,760	(100)

表 3 より，「空白」が最も検出数がい多いが，「文字列」の書き換えとともにプログラミングの理解度としては重要性は低い．プログラミングの制御構造として「for 文」と「if 文」に関する検出数が多くなっている．「while 文」は利用頻度が低いために検出数も少ないと思われる．「変数」や「数字」の書き換えの検出数が多くなっている．これは似たような問題を順次解いていくという大学の授業という性質によるものも含まれていると思われる．「式」の検出数も多くなっているが，ここに含まれる論理エラーは更なる分析が必要かもしれない．また，「その他」に分類されたものはコンパイラはエラーを出力しなかったが 2 バイト文字による修正が含まれたソースコードが検出された．

5.3　検出にかかる実行時間の評価

表 4 に示すコンピュータで図 9 に示す抽出オプションでデバッグ練習問題抽出ツールを実行した．表 5 に各年度の分析対象のソースファイルの数と分析の実行時間を示す．約 200 名の 1 年の授業の全学習履歴を一度だけ分析すれば良いということを考慮すると十分実用的な実行時間ということができる．

表 4　実測を行ったコンピュータのスペック

CPU	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
メモリ	16GB
OS	Windows 10 Pro (64 bit)

表 5　実 行 時 間

年度	ソースファイルの数	実行時間 (秒)
2017	72,193	390
2018	61,721	249
2019	69,522	272

6.　まとめと今後の課題

文法的には誤りはないためコンパイラがエラー情報を出力しない論理エラーを大量のプログラミング履歴から検出することができた．この結果を用いることで例えば for 文に関係する 1 つの論理エラーを含むソースプログラムを抽出することができる．このソースプログラムを学生に提示することで論理エラーを修正するデバッグ練習を行うことができる．

今後は，例えば変数に関する論理エラーでも，例えば二重ループに関係した変数を間違えやすかったり，繰り返しの変数と条件分岐の変数に誤りがある場合も多いと考えられる．そのようなプログラムの制御文と変数や数字の組み合わせによる分析など，さらなる詳細な分析を自動で行えるようにしたい．

謝　　辞

本研究の実施にあたり，早稲田大学理工学術院総合研究所プロジェクト研究「次世代 e-learning に関する研究」のメンバより貴重なコメントをいただき感謝いたします．本研究の一部は，独立行政法人日本学術振興会学術研究助成基金助成金基盤研究 (C)20K03082, (B)19H01721,(C)19K04914, (C)17K01101, 早稲田理工研特別勘定 1010000175806 NTT 包括協定共同研究，および，経営情報学会「ICT と教育」研究部会の助成による．

文　　献

[1] 荒本道隆，小林学，中澤真，中野美知子，後藤正幸，平澤茂一，“編集履歴可視化システムを用いた Learning Analytics～システム構成と実装,” 情報処理学会第 78 回全国大会, 横浜市, pp.4-527–528, Mar. 2016.

[2] Katsuyuki Umezawa, Manabu Kobayashi, Takashi Ishida, Makoto Nakazawa, and Shigeichi Hirasawa,“Experiment and Evaluation of Effective Grouped Flipped Classroom,” Proceeding of the 5th International Conference on Applied Computing & Information Technology (ACIT 2017), pp.71-76, July 2017.

[3] Katsuyuki Umezawa, Manabu Kobayashi, Takashi Ishida, Makoto Nakazawa, and Shigeichi Hirasawa, “Use of Student Grouping to Make Flipped Classroom More Effective,” Proceeding of the 16th Hawaii International Conference on Education, p.p. 1249-1250, Jan. 2018.
[4] 梅澤克之，小林学，石田崇，中澤真，平澤茂一，“グループ分け反転授業の実授業への適用，” 電子情報通信学会 教育工学研究会 (ET) 予稿集, pp.199-204, Feb. 2018.
[5] Katsuyuki Umezawa, Takashi Ishida, Makoto Nakazawa and Shigeichi Hirasawa, “Application and Evaluation of Grouped Flipped Classroom Method to Real Classes,” Proceeding of the International Conference on Engineering, Technology, and Applied Science (ICETA2018), p.99, June 2018.
[6] 梅澤克之，中澤真，後藤正幸，平澤茂一，“学習履歴を活用したデバッグ練習問題抽出システムの開発,” 第 18 回情報科学技術フォーラム（FIT2019）, 予稿集 Vol.3 pp.331-334, Sept. 2019.
[7] Katsuyuki Umezawa , Makoto Nakazawa, Masayuki Goto and Shigeichi Hirasawa, “Development of Debugging Exercise Extraction System using Learning History,” Proceeding of the 10th The International Conference on Technology for Education (T4E 2019), p.p.244-245, Dec. 2019.
[8] 長島和平, 長慎也, 間辺広樹, 兼宗進, 並木美太郎, “Web ブラウザを用いたプログラミング学習支援環境 Bit Arrow の設計と評価,” 情報処理学会研究報告コンピュータと教育 (CE) 予稿集, p.p.1–8, Feb. 2017.
[9] 大森隆行，丸山勝久，“開発者による編集操作に基づくソースコード変更抽出,” 情報処理学会論文誌, Vol.49, No.7, pp.2349–2359, 2008.
[10] 森一樹, 田中昂文, 橋浦弘明, 櫨山淳雄, 古宮誠一, “プログラミング演習支援のための細粒度履歴収集環境の開発,” 情報処理学会研究報告（SE）, 179(16), pp.1–6, 2013.
[11] 小林 学，後藤正幸，荒本道隆，平澤茂一，“プログラミング編集履歴可視化システムとその実践,” 日本経営工学会秋季大会予稿集，pp.8–9, Nov. 2015.
[12] 中澤 真，後藤正幸，荒本道隆，平澤茂一，“ビジュアルプログラミング言語「Scratch」のための学習履歴分析環境とその可能性－初等教育からのプログラミング教育に向けて－,” 日本経営工学会秋季大会予稿集，pp.10–11, Nov. 2015.
[13] 中野美知子，荒本道隆，吉田諭史，“プログラミング言語の学習ログ収集ソフトウェアを活用した文法矯正練習の試み，” 日本経営工学会秋季大会予稿集，pp.12–13, Nov. 2015.