差分構文木によるプログラミング授業受講者のコーディング特徴分類
Characterize Programming Class Participants using Differential Syntax Trees

青木 晃汰[1)]　　上野 秀剛[1)]
Kouta Aoki　Hidetake Uwano

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]．講義に用いられるシステムの 1 つに Online Judge System (OJS) がある．OJS は教員が提示する課題に対して，受講者が課題を解決する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．その後，教員が用意したテストケースによって採点され，各テストケースの正誤判定結果，score，コンパイル時のエラー，実行時エラーがフィードバックとして提示される．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，採点結果が満点である 100 点となるソースコードを実装することを目指す．全授業の各課題にて提出されたソースコードは受講者ごとにフォルダに分けられ特定の保存場所に貯められる．

OJS を用いた講義形態は教員が課題を直接採点する負担が軽減されており，少数の教員が多数の受講者に行う講義に適している．一方で，教員が直接採点しておらず，クラス全体の学習状況の把握を補助できるものではない．そのため，課題に対する理解が不十分な学生に対する適切なサポートができないという問題がある．そこで，各学生に対して授業内容の理解を促すためのフィードバックを自動的に生成するシステムを作成できればこの問題が解決できる．

本研究では OJS を用いた講義において学生を補助するためのフィードバック自動生成システムを作成するために，個々の学生が課題で 100 点の評価を得るまでに提出した一連のソースコードからフィードバックに有用な情報を抽出する手法を提案する．提案手法は最終バージョンとそれ以前の各バージョン間のソースコードから連続した差分の列を求め，構文木情報を付与したものを差分フローと定義する．差分フローを構成する差分は正答である最終バージョンとの差分であり，各提出段階においてどこまで正答に近づいたかを表す．そのため，個々の修正の結果が点数の増減とどう関係するか理解しやすく，受講者が正答に到達するまでのコードに対する編集行動を分析するために有用である．また，編集箇所の構文情報を含むことで，例えば for 文の条件式やブロックに対する連続した修正のような，特定の要素に対する修正と点数の増減の関係を分析することが可能であり，従来の行やスナップショットによる差分と比較して受講者の文法や学習単元に対する理解の有無の識別に有用な情報である．本論文では，2 つのソースコードを入力すると任意の形式で構文木差分を出力するツールである Gumtree[2] を用いて XML 形式で差分を出力し，そこに含まれる構文情報を抽出するシステムを開発し，複数のソースコードを対象とした出力例を示す．実装したツールで生成した差分フローから各学生のコーディング特徴を求めることで，学生自身の間違いや行動への理解を促すフィードバックの動的生成に有用と考えられる．

以下，2 章では関連研究について，3 章で OJS と抽象構文木について説明する．また，4 章で提案手法と実装ツールについて示す．5 章では，実装ツールの出力例と考察を述べる．

2　関連研究

2.1　Online Judge System

これまでに，OJS をプログラミング講義の支援に用いる研究が複数行われている [3, 4]．Hui らはプログラミングの習得に必要なプログラミング言語の理解，問題解決能力などを養うために独自のオンライン判定システムである YOJ（Youxue Online Judge）を構築した [3]．YOJ は提出されたソースコードのコンパイル・実行・正誤判定を行う Online Judge モジュール等の 7 つのモジュールで構成されており，複数のソースコードを並列処理によって実行・評価できる．Wenju らはテストケースを用いた採点では評価できないプログラムの類似性や可読性などを評価するための OJS フレームワークを提案した [4]．提案フレームワークは学生への個別フィードバック，コード品質チェック，コード類似性チェック，教育調整アドバイスの 4 つのモジュールから構成され，提出されたソースコードの総合的な解析レポートを自動的に作成し，生徒に提供する．また，採点結果を元にした統計情報を基に教師は教育スケジュールを調整できる．

これらの研究は OJS を利用したプログラミング講義という点において本研究と関連があるが，ソースコード間の差分を用いて自動的なフィードバックを生成するための研究は行われていない．本研究では提出された一連のソースコードの差分からコーディング特徴を抽出することで，受講者自身が各提出において何を誤っていたか，見落としていたか理解させるきっかけとなるフィードバックを自動生成することを目指す．

2.2　ソースコード間の差分

ソフトウェア開発などにおいて，コード間の変更履歴を理解することは重要である．これまでにオープンソースソフトウェアなどを対象にパターンマイニングやコードレビューに関する様々な研究が行われている [5, 6]．藤本らは Gumtree を拡張し，ファイルを横断するコード片の移動を検出する手法を提案した [5]．8 個のオープンソースソフトウェア (OSS) に対して提案手法を用いた実験の結果，88,848 個のコミットの中から 89,418 個のファイルを横断する移動を検出し，ファイルを横断するコード片の移動やファイル名の特徴が得られた．また既存ツールを上回る数の移動を検出した．松本らは編集スクリプトの長さが課題である Gumtree を改良し，より短く理解しやすい編集スクリプトを生成する手法を提

1）奈良工業高等専門学校情報工学科．Department of Information Engineering, National Institute of Technology (KOSEN), Nara College.

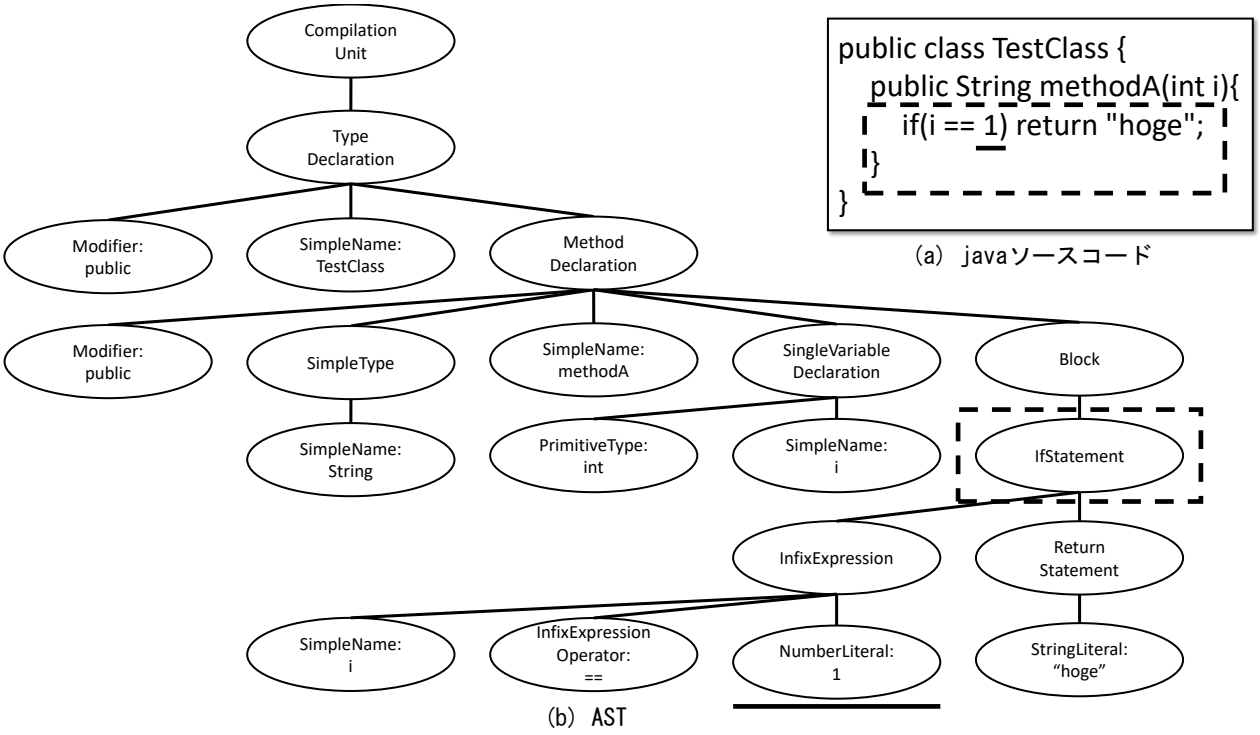


図 1: ソースコードと AST

案した [6]．7 個の OSS を対象とした実験の結果，全てのプロジェクトで編集スクリプトが短くなった．また，14 人の被験者に対する実験の結果，提案手法によって差分理解に費やす時間が減ることを確認した．

ソースコード差分に関するこれらの研究は，ソースコード間の差分の理解支援という点や抽象構文木による比較という点で本研究と関連があるが，プログラミング講義の課題に対して受講者が提出したソースコードの差分から理解を促すための研究は行われていない．本研究では講義の課題に対して提出されたソースコード間の差分から構文情報を抽出し，受講者のコーディング特徴を分析することで受講者の学習補助に利用する．

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，ある単元に対し OJS 上に提示された講義資料を元に課題を解く形式である．受講者は OJS 上に提示された課題に対応するソースコードを，自身が普段使用するエディタで作成して OJS に提出する．OJS は提出されたソースコードをプログラムの動作に必要な他のソースコードとともにコンパイル・実行し，あらかじめ用意された入力を与えたときの出力が，予期された出力と一致するか判定する．あらかじめ用意された入力と，入力に対応した正解となる出力の組をテストケースと呼ぶ．OJS は一致した出力の数に応じて点数 *score* を計算する．

受講者は提出したソースコード 1 組に対するフィードバックとして，各テストケースの正誤，*score*，コンパイルエラーの有無，実行時エラーの有無を得る．各受講者は *score* が 100 になるまでフィードバックを基にソースコードを修正，再提出する．*score* が 100 となるソースコードを提出することで課題が完了し，それまでに提出したすべてのソースコードが提出日時，*score* とともにリビジョンとして記録される．

3.2 抽象構文木 (AST)

抽象構文木 (AST: Abstract Syntax Tree) とはソースコードの構文情報を表現した木構造である．図 1 に (a)Java ソースコードと (b) ソースコードに対応する AST を示す．AST は順序木であり，各頂点がプログラムの構文上の 1 つの要素に対応した ID とラベルを持つ．表 1 に構文情報の一部を示す．親頂点と枝で結ばれた子頂点は詳細情報を表す．例えば，図 1 の下線で示した頂点”NumberLiteral:1” は図 1(a) の 3 行目 7 文字目の要素に対応しており，ラベル NumberLiteral が数値定数，値 1 がその値が 1 であることを表す. また, 図 1 の破線で示した頂点は図 1(a) の if 文全体に対応しており，ラベル IfStatement の構文情報が 2 つの子頂点 InfixExpression（if 文の条件式）および ReturnStatement（return 文）からなることを示している．

本研究の提案手法はソースコードから AST を求め，*score* が 100 となったソースコード（最終版）と最終版以外のソースコード間で AST の差分を出力する．提出された個々のソースコードを最終版と比較することで，各提出における修正が必要な箇所や，編集箇所の分布が容易に理解できる．また，各差分に構文情報を付与することで，編集内容に score や要修正というラベルを付与したデータを用いた機械学習で頻繁に間違える構文要素や処理内容を学習し，フィードバックの生成に利用できると考えられる．

4 提案手法

4.1 概要

提案手法は 1 人の受講者が 1 つの課題に対応して提出した最初のソースコードから，*score* が 100 である最後に提出されたソースコードまでの全ソースコードを一連のものとして扱うことで，正答に近づいていく修正作業の様子を表現する．図 2 に提案手法が出力する，課題に

表 1: タグと内容の対応表

構文情報	概要
ConpulationUnit	抽象構文木の root
TypeDeclaration	型宣言
Modifier	修飾子
MethodDeclaration	method 宣言
SinpleName	単純名
QualifiedName	限定名
block	method や Statement の範囲
forstatement	for 文
Ifstatement	if 文
VariableDeclarationExpression	変数宣言式
InfixExpression	中置演算式
PostfixExpression	後置演算式
NumberLiteral	数値定数

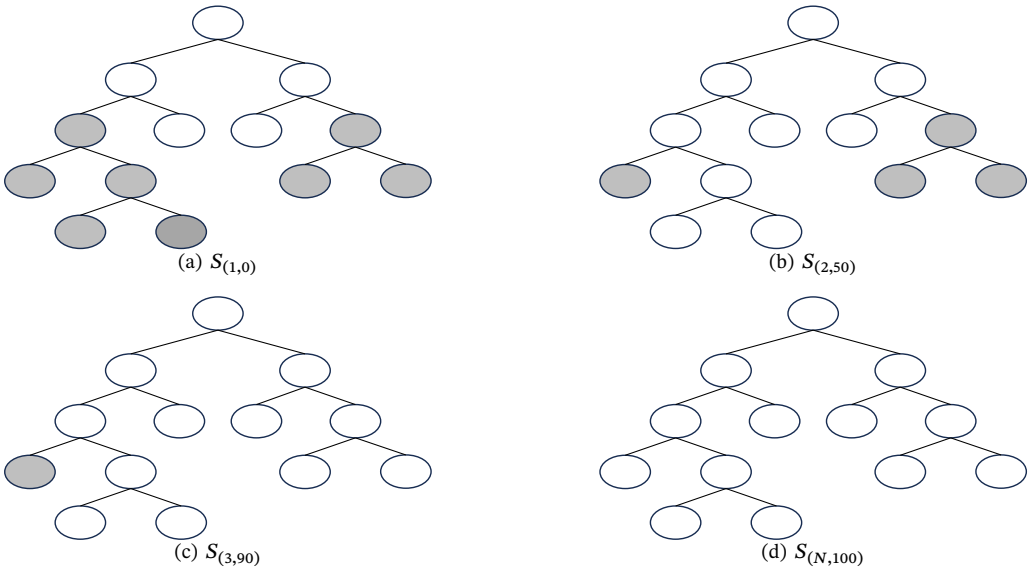


図 2: 各提出段階における構文木と差分

対して完答した際のソースコードとそれ以前の各ソースコード間の差分を示す一連の AST の例を示す．(a) から (d) の木は受講者が同じ課題に対して提出したソースコード $S_{(i,score)}$ の AST を表し，$(i = 1, 2, ..., N)$ は提出順，*score* はソースコードに対する OJS の採点結果を示す．例の (d)$S_{(N,100)}$ はある課題に対して最後に提出された *score* が 100 のソースコードであり，(a)-(c) の灰色の頂点は (d) との差分を示している．$S_{(N,100)}$ とそれ以外のコードの差分は各バージョンのソースコードのうち，*score* が 100 に満たない原因部分を表すと考えられる．受講者が提出を繰り返しながら理解を進め，正答に近づいている場合，差分は段階的に減少し，それとともに *score* が上昇する．一方で，課題の一部や特定の文法について理解が不十分な場合，対応する頂点が (d) とは異なる提出が続くため差分は減少せず，*score* は停滞，もしくは下降する．

提案手法は *score* が 100 となる最終的な提出とそれ以外のソースコード間で AST の差分を出力することで，以下の利点がある．

- 各受講者の完答を基準として $S_{(N,100)}$ に至るまでの編集の様子が理解しやすい
- 差分の構文情報から意味の抽出を機械的に行える
- 課題取り組み時の各構文に対する編集行動を表現するための情報を得られる

ソースコード間における *score* と編集箇所の変化には密接な関連があり，提案手法の出力は各受講者の理解不足箇所や誤りに気づいたタイミングを動的に識別し，必要な支援を提供するための指標になると考えられる．

4.2 実装

提案手法を用いたフィードバックシステムの概要を図 3 に示す．受講者は OJS に掲示された講義資料と課題文を元にソースコードを OJS に提出する．提出したソースコードが題意を満たさずに 100 点未満の点数が付いた場合，受講者は修正したソースコードを再提出し，100 点になるまで繰り返す．OJS システムは提出されたソースコードをそれぞれ Gumtree[2] に入力し，Ver.i$(i = 1, 2, ..., N - 1)$ と Ver.N 間の差分情報とそれぞれのバージョンの AST を出力する．Gumtree は 2 つのソースコードを入力するとそれぞれを AST に変換した上で差分を出力する．差分情報は 2 つのソースコード間

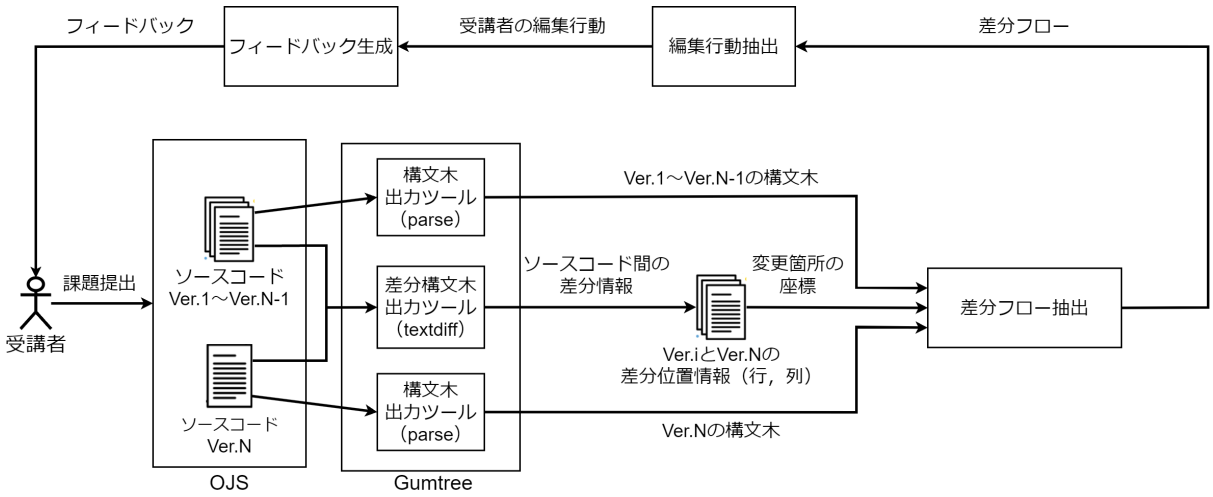


図 3: 提案手法を用いたフィードバックシステム

で行われた編集行動（挿入，更新，移動，削除）と構文情報，変数や文字，変更された範囲の座標（ソースコードの最初の文字から該当箇所までの文字数）が記されている．差分情報の座標をもとに各バージョンの AST から対応する編集箇所を抽出し，Ver.1 と Ver.N の差分とする．提案手法は上記の処理を最終バージョンを除いた全てのバージョンに対して実行し，その集合を差分フローとする．差分フロー抽出の処理は Python で作成し，XML 形式で出力される．

5 結果・考察

5.1 実装システムの出力例

図 4 に編集前と変更後の各ソースコードと AST を示す．図 4(b)(d) の構文木について，簡単化のため一部の子要素を省略している．図 4 のソースコード (a)(c) 間では 3 行目の for 文の終了判定式が”i>10” から”i<10” に編集されている．対応する AST に着目すると最下段中央の頂点に差があり，差分が中置演算式の演算子”InfixExpression Operator” にあることを示している．また，差分の親要素に着目すると順に中置演算式”Infix Expression”，for 文”ForStatement”，ブロック”Block”，method 宣言”Method Declaration” と続いており，差分のある箇所がメソッド内の for 文の中置演算式の一部であることを示している．提案手法が出力する差分は 2 ファイル間で差分のあった文字だけではなく，構文要素に着目した，編集箇所を含む親要素を抽出する．そのため，ある 1 つの編集を演算子，中置演算式，for 文，メソッドに対する編集として表現することで異なる粒度に基づいた編集行動の分析を容易にする．

図 5 に図 4 に示した 2 つのソースコード間の差分を提案手法で出力した例を示す．図 (a) の AST は変更前のソースコードを基準として，編集された文字の構文要素を表すノードを灰色で表示し，すべての親要素のノードを強調表示している．図 (b) は同じ差分を XML で表現しており，パーサなどを用いることで機械的な処理が容易である．ソースコード間の差分を構文要素の集合として捉えることで，連続した差分（差分フロー）の特徴を抽出でき，特徴に合わせたフィードバックの自動生成が可能となる．

5.2 差分フローを用いた編集行動の理解

提案手法の出力を用いた，受講者の編集行動の分析例を示す．図 6 に奈良高専 3 年生を対象としたプログラミングの講義において，1 人の受講者がある課題に対して提出したソースコード群を示す．図はそれぞれ左側に (a)1 回目，(b)5 回目，(c)7 回目，(d)9 回目，(e)10 回目に提出されたソースコード，右側に最後に提出されたソースコード ($S_{11,100}$) を表しており，差分が強調されている．また，紙面の都合上，一部のメソッドのみを表示している．本課題は通貨を表し，メソッドによって日本円とアメリカドルを変換可能な Money クラスを作成する内容で，フィールド変数 3 個とメソッド 6 個が含まれる．対象の学生は 11 回の提出で 100 点に到達し，ソースコードの SLOC は 25 行だった．対象の学生が提出したソースコードを見ると，大部分の提出において指定したレートを基に通貨の両替を行う “exchange メソッド” を編集している．1 回目の提出（図 6(a)）の時点で exchange メソッドを正しく実装できているが，フィールド変数である rate の初期化を行っていないため exchange メソッドで 0 除算が発生し，評価が 0 点になっている．その後，対象の学生は 4 回目の提出まで一部の変数名を修正しただけのほぼ同じソースコードを提出しており点数に変化はない．5 回目の提出（図 6(b)）で別のメソッドを修正し点数が 50 点に上昇した後，7～10 回目（図 6(c)(d), 図 7）でエラーが発生している exchange メソッドを変更しているが，処理内容は変わっておらず点数に変化はない．最後の提出で rate の初期化を追加し，点数が 100 点に到達している．

図 8 に図 6 のソースコードに対応する AST の組を示す．ソースコードの場合と同様に，左側に (a)1 回目，(b)5 回目，(c)7 回目，(d)9 回目，(e)10 回目に提出されたソースコードの AST，右側に最後に提出されたソースコードの AST を表している．色の濃い頂点は差分取得対象のソースコード (各図の左側の場合，$S_{11,100}$) と比較して挿入，更新，移動，削除の差分があることを表す．また，実線で囲まれた頂点群が exchange メソッドに対応し，点線で囲まれてた頂点が rate フィールドに対応する．

図は編集履歴の大部分が exchange メソッドに集中しており，課題が進むにつれて最終版のソースコードとの

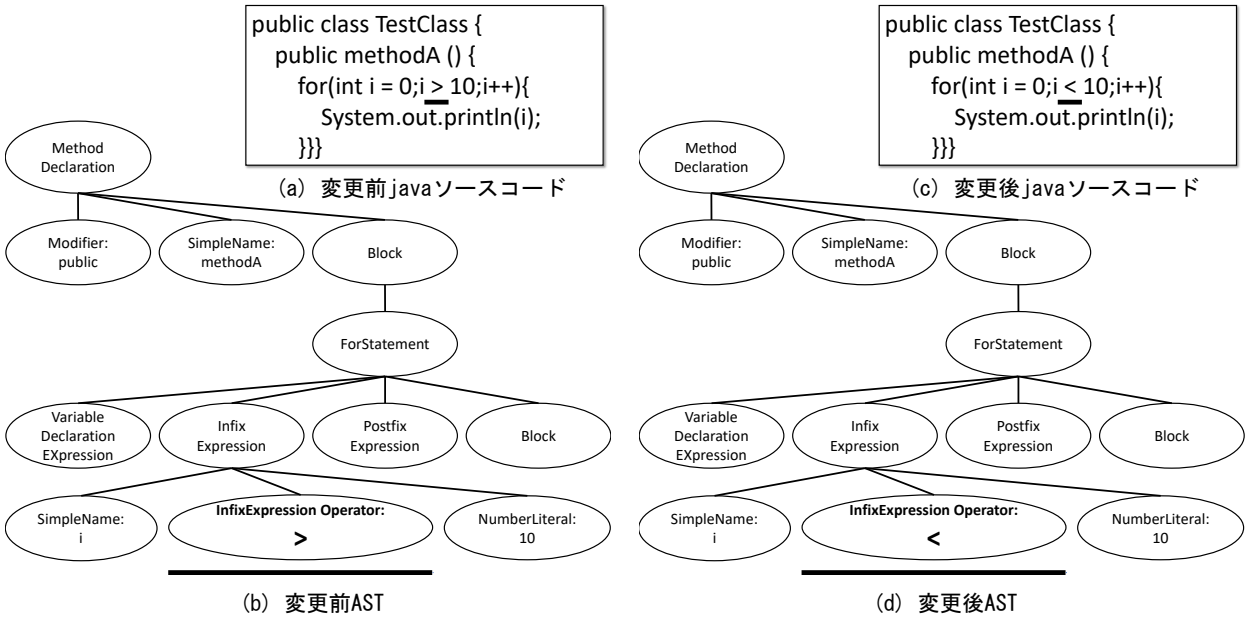


図 4: 変更前後のソースコードと AST

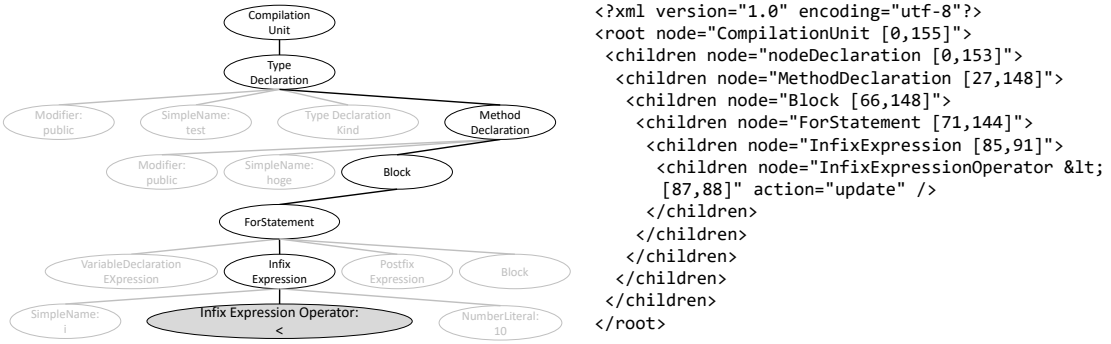


図 5: ソースコード間の差分

差分が減少していることを示す．一方で，点線で囲まれた rate フィールドについては提出 1 回目から誤っているにもかかわらず，10 回目の提出まで編集されておらず（図 8(e)），誤りに気づいていない．図に示した差分フローは rate フィールドに対する誤りが提出 1 回目から継続して存在していることを示しており，XML で出力された差分フローからその特徴を抽出することができる．提案手法が出力する差分フローは最終提出のソースコードと比較してどの構文要素に差分があるかを表すため，一連の提出の中でどのメソッドやブロック，変数に編集作業が集中しているか，その遷移を容易に読み取ることができる．

6 おわりに

本研究では OJS を用いたプログラミング講義において 1 人の受講者がある課題に完答するまでに提出した一連のソースコード群を対象として，最終版のソースコードとそれ以前に提出されたソースコードから差分の構文木を生成することで編集行動の分析に有用な差分フローを出力する手法を提案した．差分フローは完答のソースコードとの差分であるため，差分の変化を分析することでプログラム中のどの部分の編集を繰り返しているのか機械的な処理で得ることができる．また，差分フローは構文要素の集合として表現されるため，複数の箇所で特定の要素（例えば for 文）に関連した編集を行っている様な編集行動が抽出でき，受講者の苦手とする要素の検出に有用と考えられる．

本研究の今後の発展として以下の要素が挙げられる．

- 差分フローから受講者の編集行動の特徴を抽出する手法の開発
- 差分フローから抽出した受講者の苦手要素に対応したフィードバックの自動生成と提示

7 参考文献

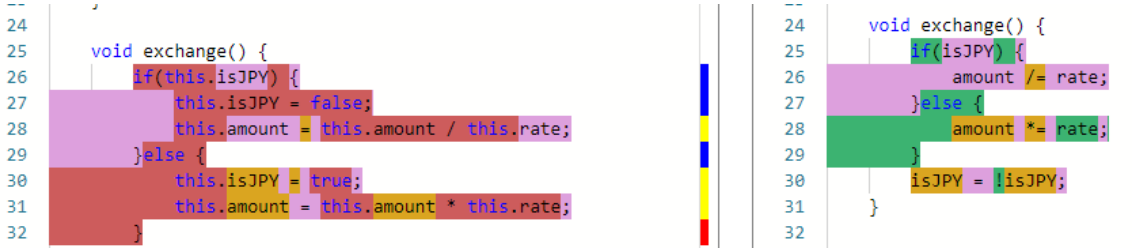
参考文献

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

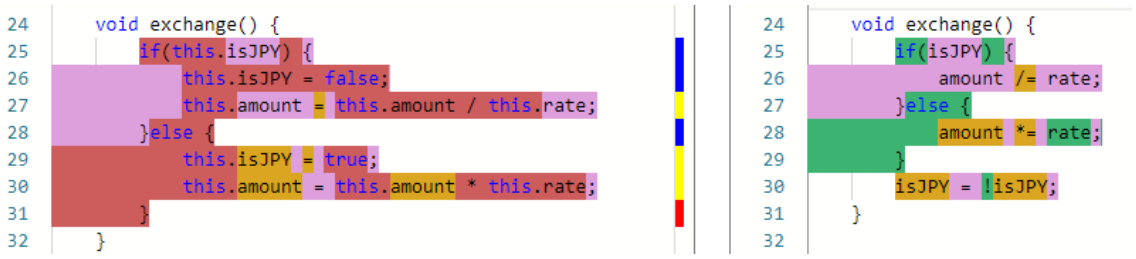
[2] J. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, ”Fine-grained and accurate source code differencing. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE2014), pp.313-324, 2014.

[3] H. Sun, B. Li, M. Jiao, ” YOJ: An online judge system de-

signed for programming courses”, In Proceedings of the 9th International Conference on Computer Science Education (ICCSE2014), pp.812-816, 2014.
[4] W. Zhou, Y. Pan, Y. Zhou, G. Sun, ”The framework of a new online judge system for programming education”, In Proceedings of ACM Turing Celebration Conference (TURC2018), pp.9-14, 2018.
[5] 藤本章良, 肥後芳樹, 松本淳之介, 楠本真二, プロジェクト全体の抽象構文木構築によるファイル間の移動コード検出, 電子情報通信学会論文誌 D, Vol.J104-D, No.4, pp.242-254, 2021.
[6] 松本淳之介, 肥後芳樹, 楠本真二, より短い編集スクリプトを目指して— 行単位の差分情報に基づく GumTree の拡張, 電子情報通信学会論文誌 D, Vol.J103-D, No.8, pp.579-590, 2020.



(a) $S_{1,0}$



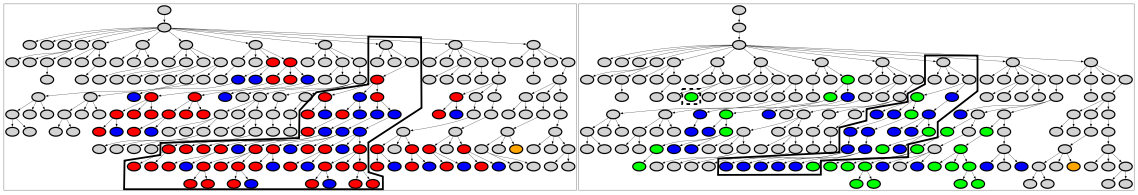
(b) $S_{5,50}$

(c) $S_{7,50}$

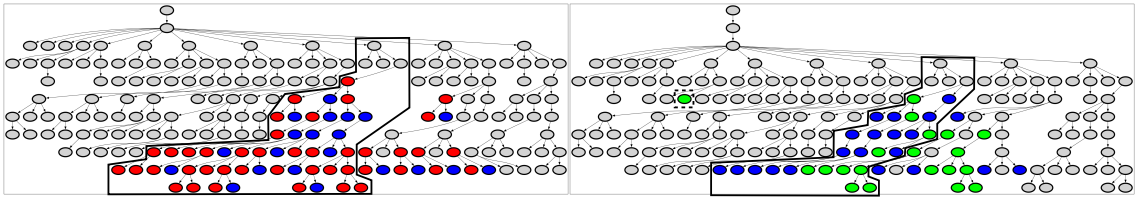
(d) $S_{9,50}$

図 6: exchange メソッドの変更履歴

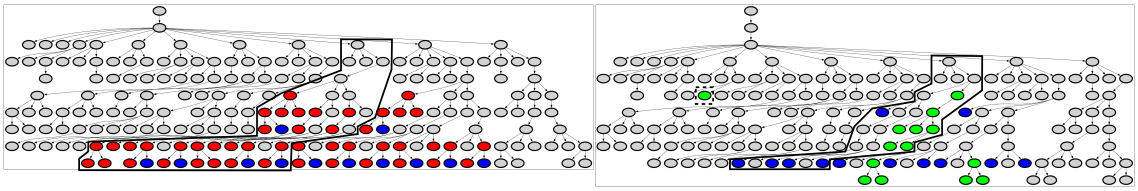
図 7: クラスフィールドの変更履歴 ($S_{10,50}$ と $S_{11,100}$)



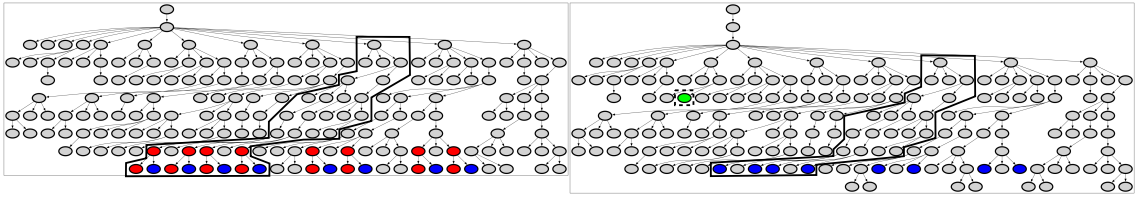
(a) $S_{1,0}$



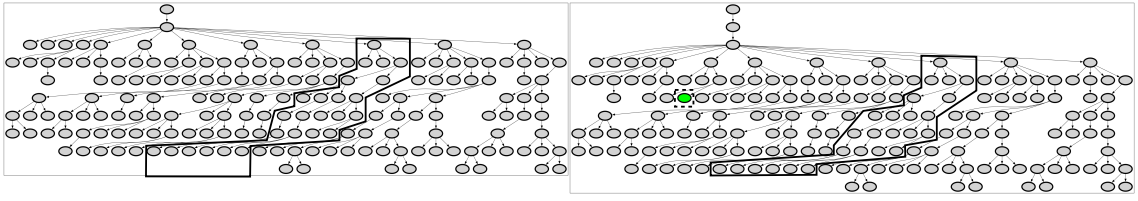
(b) $S_{5,50}$



(c) $S_{7,50}$



(d) $S_{9,50}$



(e) $S_{10,50}$

図 8: Money.java の AST(一部)