卒　業　研　究　報　告　書

令和4年度

研究題目

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

指導教員　　上　野　秀　剛　　准教授

氏　　名　　青　木　晃　汰

令和05年01月26日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

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

上野研究室　青木晃汰

本校におけるプログラミング講義の一部ではオンラインで課題のソースコードを提出し，システムが自動採点する方式がとられている．この授業形態は少数の教員で受講者が多数いる1クラスに対して行う講義に適したものである．しかし，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ため受講者個人に合わせた適切なサポートができないという問題がある．そこで，受講者個人が課題に対して提出した一連のソースコードから得た情報を用いて，受講者自身がコーディングの誤りに気づけるような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前述した問題を解消できると考えられる．本研究では，OJSを導入した授業で1人の受講者が取り組んだある1つ課題に対して提出された複数のソースコードから，受講者の編集履歴を表す差分を出力する手法を提案する．提案手法は提出された全てのソースコードから構文木を出力し，100点を取得したソースコードとそれ以前に提出された複数のソースコードの差分を求め，連続した差分の列として出力する．本研究では，差分の列を差分フローと定義する．編集履歴を対象とした研究の多くは，あるバージョンのソースコード（バージョンN）とその直前のバージョン（バージョンN-1）の差分を用いている．提案手法は100点を取得した最終バージョンとそれ以外の差分を用いることで各提出段階のソースコードにおいて100点を取得するために編集が必要な箇所を明確にすることが出来る．また，提案手法が出力する差分フローはソースコードのテキストの差分ではなく構文木情報を用いることで編集箇所に加えて親要素の情報も含んだ差分情報として出力される．そのため，異なる行に対する編集を同一の構文や要素（メソッドやクラスなど）に対する編集として扱えるため，ソースコード内の各構文に対する編集行動を表現することが出来る．本研究では提案手法を実装し，OJSを用いた講義で提出された受講者のソースコードを対象にその有効性を確認する．提案手法に必要な処理を実装するために，1)差分フローに含まれる各変更箇所の座標から変更箇所を特定するツール，2)変更箇所の親要素を取得するツールの2つを実装する．本研究の成果として，差分に含まれる構文情報から，機械的処理に必要な特徴や受講者の編集行動を表現する情報を得られることが確認できた．提案手法が出力する差分フローを分析することで，差分箇所の増減から各受講者の課題中の理解不十分な箇所や特定の要素に対する意識の抜け落ちを読み取れることや，複数の誤りが混在するソースコードに対しても受講者の誤りの特徴が明確になることが期待される．本研究の発展として,「移動」情報によるノイズを削除，フィードバック作成する際に「移動」情報を省いた編集行動の情報で構成することと編集箇所と同じ親を持つ構文情報を取得することで式を構成する字句に関連性を持たせることが考えられる．

目次

1 はじめに

現在，本校におけるプログラミング講義の一部は，資料による座学を行った後，Online Judge System (OJS) での演習という授業形態で行われている．講義で用いられている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全員に学習の補助を行う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，学生個人が課題に取り組み始めてから100点の回答を得るまでに提出された一連のソースコードを分析して，学生への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，差分フローと定義する．差分フローは従来の行やスナップショットによる差分を用いた分析と違い，コードに対する受講者の編集行動を示す情報を得られる．また，差分フローに構文木情報を付与することによって，それらの親要素から機械的処理に必要な構文情報や，ソースコード内の各構文に対する編集行動を示す構文情報を得られる．

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[1]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分析を行う．分析結果から1つの課題を通して編集した箇所の構文情報を得ることが出来れば，各学生のコーディング特徴の分類が可能になる．

以下，2章では関連研究について，3章でOJSを用いた講義や研究に用いる理論である抽象構文木について説明する，また，4章で最終的なシステム像と本研究で実装した部分について示す．5章では，研究の過程で得た結果を基に考察を述べる．

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[2, 3]．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[2]．YOJは提出された解答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いる．それぞれのモジュールを拡張することによって，並列処理による実行・評価を行うことが出来る．Wenjuらは従来のOJSのテストケースによる採点のみの評価方法でプログラムの品質を評価しないなどの欠点を解決するため新しいOJSのフレームワークを提案した[3]．このシステム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ている．

OJSを対象としたこれらの研究は，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，自動採点を行った際のフィードバックとソースコード間の差分を用いた分析を合わせた研究は行われていない．本研究では課題に取り組んだ際に提出された一連のソースコードから差分を出力し，コーディング特徴を抽出する．そこから得た情報を用いて間違いや行動に気づけるようなフィードバックを生成するシステムを作成することが出来れば，教師の負担が軽減されると共に，受講者の学習効率が上がると考えられる．

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[4, 5]．藤本らはGumtreeを拡張し，ファイルを横断するソースコードの移動を検出する手法を提案した[4]．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合計で89,418個のファイルを横断する移動を検出でき，ファイルを横断するソースコードの移動やファイル名に幾つかの特徴が得られた．また既存ツールと比較を行った結果，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，より短く理解のしやすい編集スクリプトを生成する新たな手法を提案した[5]．提案手法を評価するために7個のOSSで実験を行い，全てのプロジェクトで編集スクリプトの長さを短くすることに成功した．また，提案手法が出力する編集スクリプトが理解の助けになることを14人の被験者に対して実験を行い，差分理解に費やす時間が全体として減ることを確認できた．

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における課題において各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て提出された一連のソースコード間の差分から提出間の編集箇所の構文情報を抽出することによって，受講者のコーディング特徴を分析し，受講者の学習補助に役立てることが出来ると考えられる．

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，ある単元に対して座学の後に課題を解く形式である[7]. 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コンパイル・実行し，実行結果がテストケースと一致するか判定する．テストケースとはプログラムが要件を満たしているかテストするもので，プログラムに対する入力と，入力に対して望まれる出力の組みを表す. 各課題に対するテストケースは教員によって事前に用意されOJS上に保存されるとともに，テストケースの一部が課題とともに受講者に提示される．OJSは受講者が提出したプログラムに対して，テストケースとして用意された入力を渡し，出力される標準出力の内容がテストケースの出力と一致した数に応じて以下の式で点数scoreを計算する．

$$score = \frac{\text{テストケース正解数}}{\text{テストケース数}} \times 100 \tag{3.1.1}$$

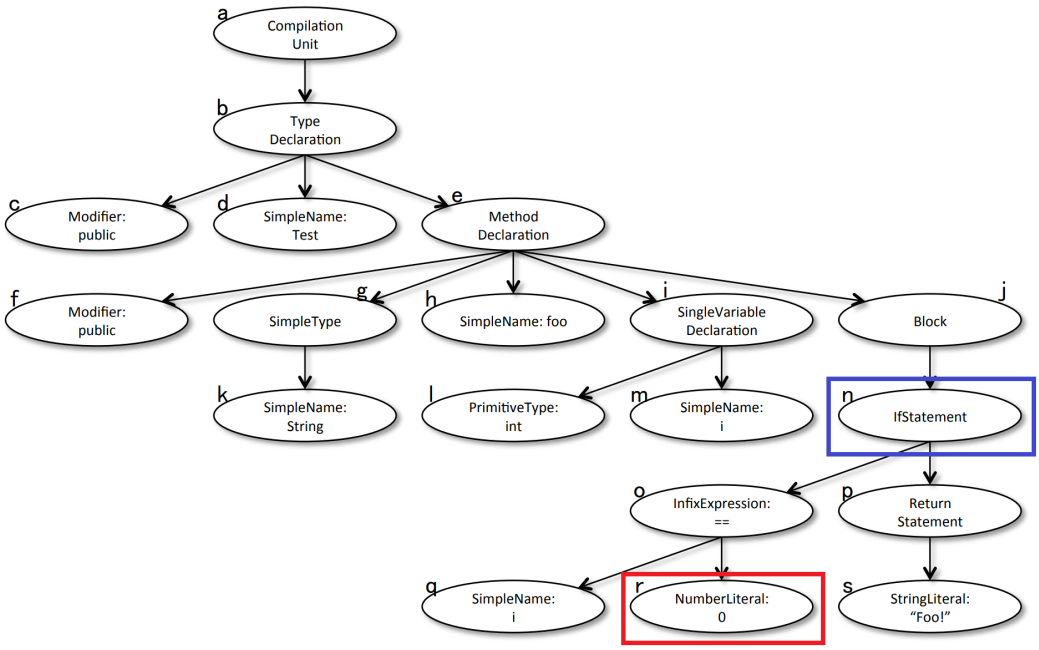
受講者は提出したソースコード1組に対して，各テストケースの正誤判定結果，score, コンパイルエラーの有無，実行時エラーの有無を得る．各受講者はscoreが100になるまで，ソースコードを提出して得たフィードバックを基に任意個所を修正し，再提出するという作業を繰り返す. scoreが100となるソースコードを提出することで課題が完了し，それまでに提出したすべてのソースコードは提出ごとにリビジョンとして，提出日時，scoreと共に記録される[7].

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[6]. 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. このASTはプログラムの構造に対応する19の頂点を持つ．ASTは順序木であり，子頂点の数に制限はない．AST上の頂点は構文上の1つの要素を表し，枝で直接結ばれた子頂点はその詳細情報を表す. それぞれの頂点はIDとソースコードの要素に対応するラベル，ソースコード内の実際のトークンに対応する値を持つ．例えば，図1(b)の赤で示した頂点の「NumberLiteral:0」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0がその値が0であることを表す. また，図1(b)の青で示した頂点は図1(a)のif文の要素に対応しており，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．

```
public class Test{
 public String foo(int i){
   if(i == 0) return "Foo!";
 }
}
```

(a) ソースコード



(b) AST

図 1 ソースコードと AST の例

本研究が提案する手法は一連のソースコードに対して AST を求め，score が 100 となったソースコードとそれ以外のソースコード間で AST の差分を出力する．この差分に含まれる情報には構文情報が含まれている．ソースコード間の差分を構文情報で表すことで，差分が持つ意味の抽出を機械的に行えるようになり，差分から得た受講者のコーディング特徴から自動的にフィードバックを生成できると考えられる [6].

4 提案手法

4.1 概要

提案手法はある1人の受講者が1つの課題に対応するソースコードを最初に提出した時から，scoreが100であるソースコードが提出されるまでに記録されたすべての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．OJSを用いた講義において提出されるソースコードのscoreは変動するため，各ソースコード間の編集箇所とscoreの変化には密接な関連がある．例として，for文のループ処理を10回行い，1ループごとに“i=”[iの値]と“ABC”という2行を出力するプログラムを作成するという課題に対して，提出された一連のソースコードを図2に示す．図の四角は提出されたソースコードとその採点結果(score)，矢印はバージョンの遷移を示す．また，ソースコード中の下線は直前に提出されたソースコードからの差分を示す．図が示す一連の提出は，Ver.2で2行目のprintln文の出力フォーマット，Ver.3で3行目の出力文字abcの大文字化，Ver.4で再び2行目のフォーマット修正とif文の条件式を修正し，Ver.4で正答（scoreが100点）に到達している．

隣接したバージョン間(Ver.1→Ver.2, Ver.2→Ver.3, Ver.3→Ver.4)での差分を表1に，scoreが100である最終版(Ver.4)とそれ以外の各バージョン間(Ver.1→Ver.4, Ver.2→Ver.4, Ver.3→Ver.4)の差分を表2に示す．表1の隣接するバージョン間の差分からは提出ごとにどのような変更があったか確認できる．一方で，個々の変更内容が課題の解答として誤っている可能性があり，差分情報からその編集内容の正しさを評価することが困難である．例えば，Ver.2で修正された2行目のprintln文は修正後も不正解であり，Ver.4で再び修正されている．連続したバージョン間の差分はその変更内容が後のバージョンで再度修正され削除される可能性があるため，以降のバージョンを確認して各差分が最終バージョンと同のような関係にあるか理解する必要がある．

それに対して，表2の最終版との差分は正しい出力を行う最終版のソースコードとの差分であるため，差分情報は各バージョンにおいてscoreを100にするために必要な編集箇所を示す．例えば，Ver.2で修正された2行目のprintln文は修正後も不正解であるため，Ver.2→Ver.4の差分に2行目が残っており，後のバージョンで再度修正されていることが分かる．表2の各行を見ると，Ver.1→Ver.4の差分とVer.2→Ver.4の差分はいずれも3行の差分が出力されており，Ver.2で行った修正が不十分であることを表す．一方で，Ver.3→Ver.4の修正で3行目が差分から消えており，Ver.4で行った3行目に対する修正が正答に近づくための修正であることが分かる．提案手法はプログラミング講義の課題取り組み中に学生が提出した一連のソースコードから表2のような差分を作成することで，課題取り組み開始から完了までの受講者の誤りの内容と編集すべき箇所の変化を教員や学生に提示する．ま

表 1 隣接したバージョン間の差分

対象	差分
Ver.1 → Ver.2	挿入 2 行目：[“i” +]
Ver.2 → Ver.3	更新 3 行目：[“abc”] → [“ABC”]
Ver.3 → Ver.4	更新 2 行目：[“i”] → [“i = ”] 更新 5 行目：[i == 10] → [i == 9]

た，差分フローは一連の最終バージョンとの差分を出力するため，どの時点で混入した不具合がどのタイミングで除去されたのかを容易に理解できる．表2の例では，Ver.1の時点で3箇所の不具合が混入しており，3行目（“ABC”の表示文）がVer.3まで，2行目（iの表示文）と5行目（ifの条件式）がVer.4まで残っていることが分かる．2行目の不具合はVer.2で一度修正されていることから，不具合の存在には気づいた一方で，仕様を正しく理解しておらず，Ver.4まで解決できなかったことを示唆する．提案手法はバージョンごとの差分を受講者の一連の修正として出力することで受講者の理解不足の解消や気づきのタイミングを識別可能であり，異なる課題に対する傾向を分析することで，気づきが遅れる／理解が不足している要素の抽出が可能と考えられる．

提案手法の概要図を図3に示す．受講者が課題の取り組み中に提出した一連のソースコードの各バージョンをVer.1〜Nとする．score100をVer.N，score0〜99をVer.1

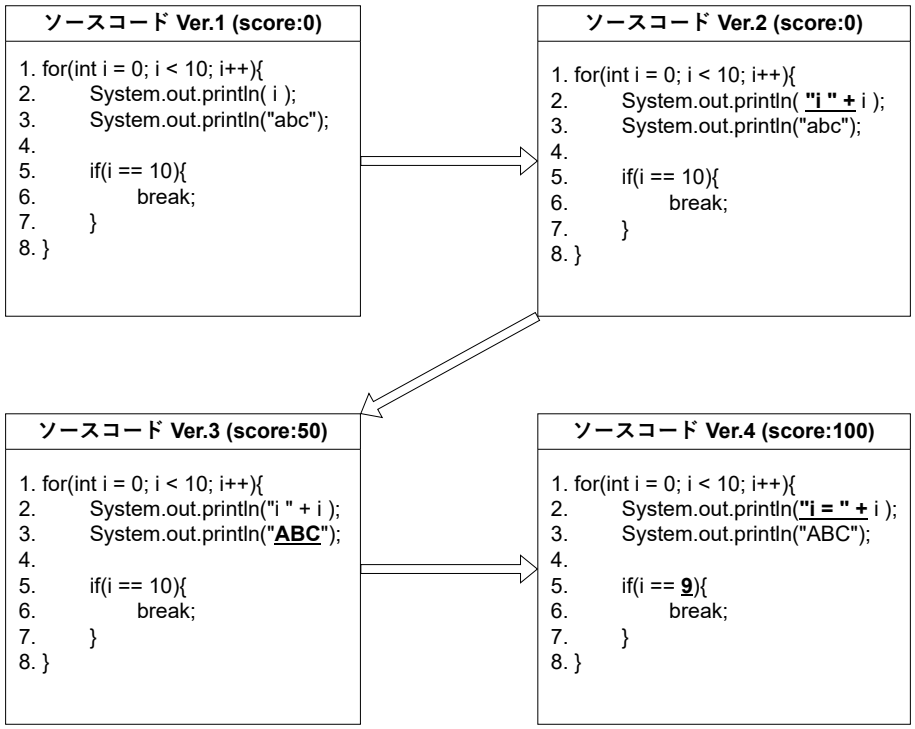


図 2 一連のソースコード編集

表 2 最終バージョンとそれ以外のバージョン間の差分

対象	差分
Ver.1 → Ver.4	2 行目：[i] → [“ i = ” + i] 3 行目：[“ abc ”] → [“ ABC ”] 5 行目：[i == 10] → [i == 9]
Ver.2 → Ver.4	2 行目：[“ i ” + i] → [“ i = ” + i] 3 行目：[“ abc ”] → [“ ABC ”] 5 行目：[i == 10] → [i == 9]
Ver.3 → Ver.4	2 行目：[“ i ” + i] → [“ i = ” + i] 5 行目：[i == 10] → [i == 9]

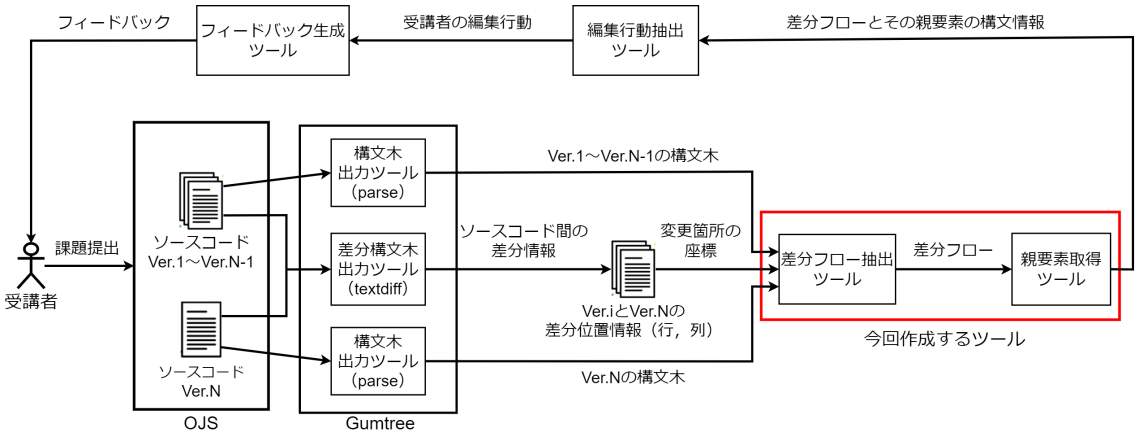


図 3 提案手法　概要図

～N-1 とし，差分を出力する際は Ver.i(i = 1～N-1) と Ver.N の二つのソースコードを入力とする．

図3のプロセスは以下の通りである.

1. 教師が課題を提示し, 受講者の課題への取り組みが完了
2. 課題提出履歴からGumtreeにVer.Nのソースコードと Ver.1〜N-1の各ソースコードを入力し, それぞれのASTを比較してその差分をXML形式で出力する
3. 差分情報を含むXMLファイルから変更前後の各ソースコードの変更箇所の座標を抽出する.
4. 座標から変更箇所を特定し, その箇所の構文情報を抽出することで, 差分のフローを抽出する.
5. 各変更箇所の親要素を辿り, それぞれの構文情報を変更箇所に近い順に取得する.
6. 保存した親要素の構文情報から変更箇所がどの構文に属していたかを確認する.
7. 課題取り組み開始から完了までの変更箇所の構文情報をまとめ, 課題を通した受講者のソースコードに対する編集行動を抽出する.
8. 編集行動からコーディング特徴を確認できるフィードバックを生成する.

今回はこの内, 図3の赤枠で囲われた箇所の差分のフローに含まれる各変更箇所の座標から変更箇所を特定するツールとそれらの親要素を取得するツールの作成を行う.

4.2 Gumtreeを用いた差分出力

今回のシステムでは，差分を出力する際にGumtree[1]を用いる．Gumtreeは入力として編集前後の二つのソースコードを入力することで，それぞれASTに変換し，出力として任意の形式で構文木差分を出力する. 差分を出力する際にASTを用いるため，構文情報が含まれた木構造で差分が得られる．これによって単なる行やテキストの差分とは異なり，差分として出力された箇所がプログラムにおいて構文上どのような役割を担っていたかを機械的に処理することが可能になる．このツールはテキスト，ブラウザ，木グラフなど様々な形式で差分の出力が可能であるが，今回は機械的処理を行うため，テキストをXML形式に変換して出力する．差分を出力する際に用いるコマンドと，ソースコードの木構造を出力するコマンドを以下に示す.

差分出力

gumtree textdiff /diff/left/Main.java /diff/right/Main.java -f XML -o /diff/left/diff.xml

木構造出力

gumtree parse /diff/left/Main.java -f JSON -o /diff/left/tree.json

木構造出力コマンドは一度JSON形式で出力してから作成するシステムでXML形式に変換するとタグ内にテキストとして必要な情報が格納されるため出力する形式をJSONファイルとしている．最終版のソースコードの各提出時点のソースコードとの差異を確認し，Ver.Nのソースコードが完成するまでの受講者の編集の流れを確認するため，GumtreeにはVer.Nのソースコードと Ver.1〜N-1の各ソースコードが対になるように入力する．ある生徒の一つの課題の差分を木構造として出力した例を図4に示す. 各ノードには本来「NumberLiteral: 1 [100,101]」のように構文情報とコード中の文字，その座標が記されているが，構文木全体の変化を確認しやすくするために各ノードを空白で表示している．各図の左はVer.1〜N-1の構文木，右はVer.Nの構文木を表す. また，ノードの各色とその編集内容の対応を表3に示す.

図4の変更があった各ノードを確認すると，課題の取り組みが進むにつれて変更箇所が少なくなっており，それに伴って差分情報を含むXMLファイルの変更情報の数も少なくなっていることが確認できる．

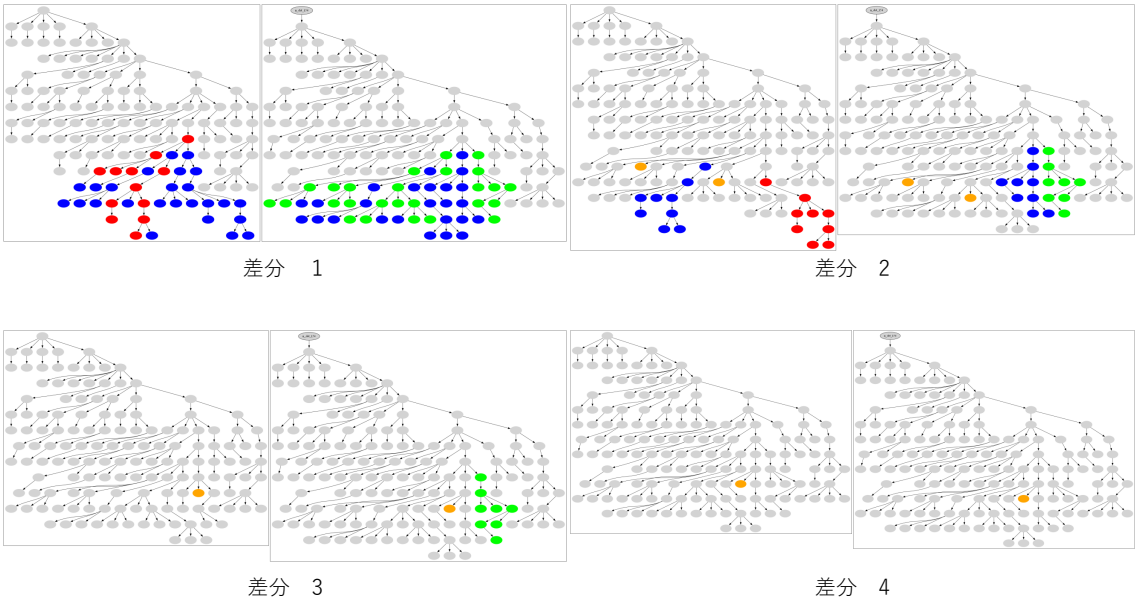


図 4 ソースコード差分構文木　例

表 3 ノードの色と編集内容

色	編集内容
灰色	変化なし
緑色	挿入
橙色	更新
青色 （ブラウザ表示の時は紫色）	移動
赤色	削除

4.3 変更箇所の特定

差分情報が含まれるXMLファイルに記された変更箇所の座標から編集前後のソースコードの該当箇所を特定するツールを作成する．これを用いることで，差分情報を含むXMLファイルの編集箇所の座標から，編集された箇所のソースコード内での構文情報を得ることが出来る．言語はPythonを用いた．変更箇所の座標情報が含まれる差分情報を含むXMLファイルの内容を一部抜粋して図5に示す．

actionタグ内にはソースコード間で行われた編集行動の情報が含まれている．各タグ名は挿入，更新，移動，削除を表しており，それと共に構文情報，変数や文字，変更された範囲の座標が記されている．座標はソースコードの最初の文字から該当箇所までの文字数で表されている．今回のツールでは，各変更箇所の先頭の座標と取得したい情報が含まれるタグ名を入力することによって，変更箇所が含まれる編集前後どちらか一つのソースコードから該当箇所の必要な情報を出力する．タグと格納されている情報の対応を表4に示す．

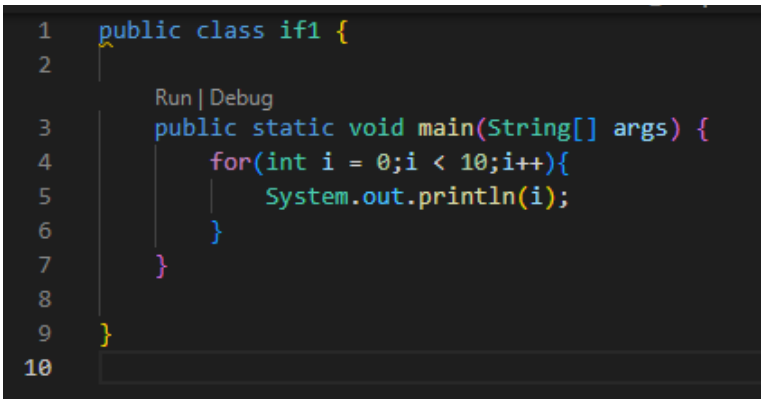
```
<actions>
  <update-node tree="SimpleName: NUM [140,143]" label="num"/>
  <move-tree tree="METHOD_INVOCATION_RECEIVER [192,202]" parent="MethodInvocation [164,187]" at="0"/>
  <update-node tree="SimpleName: NUM [181,184]" label="num"/>
  <update-node tree="NumberLiteral: 3 [185,186]" label="2"/>
  <delete-tree tree="METHOD_INVOCATION_RECEIVER [164,174]"/>
  <delete-node tree="SimpleName: print [203,208]"/>
  <delete-tree tree="METHOD_INVOCATION_ARGUMENTS [209,214]"/>
  <delete-node tree="MethodInvocation [192,215]"/>
  <delete-node tree="ExpressionStatement [192,216]"/>
</actions>
```

図5 差分1　変更箇所

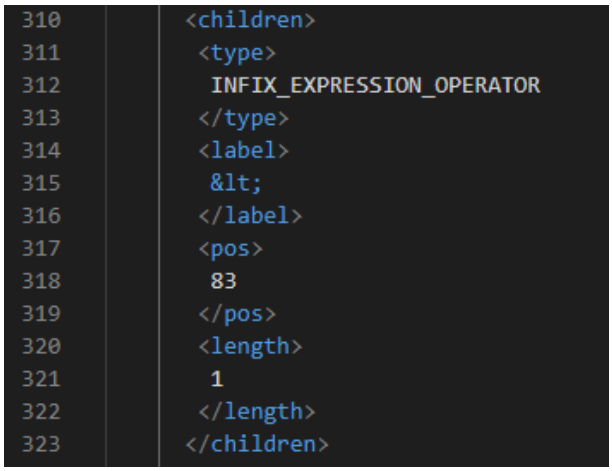
表4 タグと内容の対応

タグ名	内容
type	構文情報
pos	座標
label	変数・数値・文字
length	該当範囲

編集前のソースコードを木構造にして出力したXMLファイルの一部を図6に示す. 図6より, 図の赤枠で示した箇所の「＜」を表す情報がXML形式に変換されている. typeタグに中置演算子を表す「INFIX_EXPRESSION_OPERATOR」, labelタグに「＜: less-than sign」を表す「<」, posタグにその座標とlengthタグに1文字であることを示す「1」がテキストとして構文情報や座標などが格納されていることが確認できる.



(a) ソースコード



(b) XMLファイル　一部抜粋

図6 編集前ソースコード

図6aのソースコードからfor文の条件式の内容を変更したソースコードとその間の差分情報を含むXMLファイルの内容を図7に示す．図7aから，クラス名「if1」と，for文の条件式の「＜」が橙色で示されており，それぞれ「if2」と「＜＝」に更新されていることが確認できる．

今回のシステムでは，必要な情報を特定する際，取得する対象のタグ名としてtypeを入力として与える．今回の手法では，Python上でxmlを扱うためのモジュールであるElementTreeを用いてxmlへの操作を行う．まず，ElementTreeモジュールを使ってxmlファイルを木構造に変換する．変換した木構造の根を辿り，ファイル全体のposタグの内容を確認し，入力した座標と一致するposタグと同じ高さにあるtypeタグの内容を得る．posタグの内容は，for文などの構文や条件式などの式の最初の座標を格納する場合，図8のように複数個所で同じ値になる可能性がある．その場合，より根の末端に近い情報を得るものとする．

if1.java

```
1  public class if1 {
2
3      public static void main(String[] args) {
4          for(int i = 0;i < 10;i++){
5              System.out.println(i);
6          }
7      }
8
9  }
10
```

if1.java

```
1  public class if2 {
2
3      public static void main(String[] args) {
4          for(int i = 0;i <= 10;i++){
5              System.out.println(i);
6          }
7      }
8
9  }
10
```

(a) ソースコード

```
42  <actions>
43    <update-node tree="SimpleName: if1 [13,16]" label="if2"/>
44    <update-node tree="INFIX_EXPRESSION_OPERATOR: &lt; [83,84]" label="&lt;="/>
45  </actions>
```

(b) 差分情報　一部抜粋

図7 ソースコード編集　例

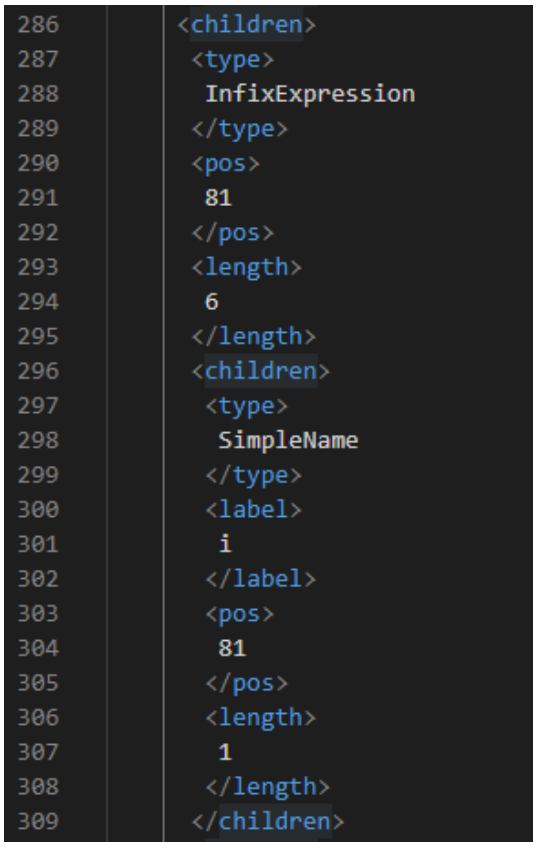


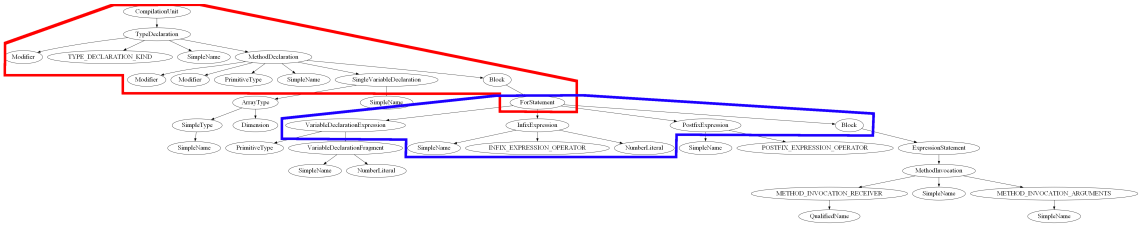
図 8 複数個所の pos タグの同一座標　例

4.4 変更箇所の親要素の取得

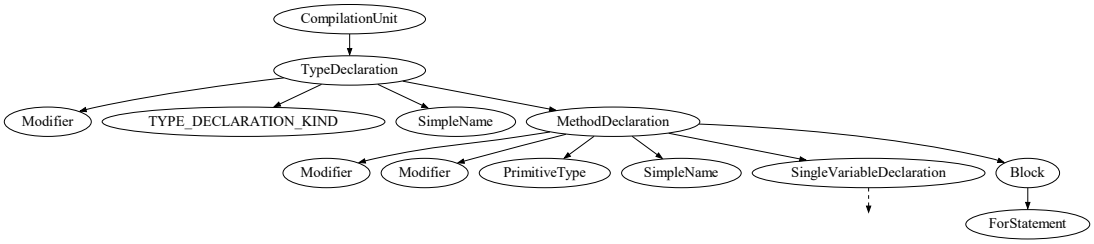
ソースコードの差分情報から受講者の編集行動からフィードバックを生成するにあたって，ソースコード中の構文のどの要素に対して変更を行っているかでフィードバックの内容は変わる．例えば，for文の内容が編集されていた場合では，条件式とループの関係が理解できてないなどの原因が考えられるが，if文の内容編集されていた場合では，条件式と分岐の関係が理解できていないことが考えられる．また，for文の内，条件式の内容とループの内容のどちらかが編集されていた場合でも同様に，受講者の理解できていないと考えられる点が異なる可能性がある．そのため，有効なフィードバックを生成するには，編集箇所がソースコード中でどのような構文に属していたかの情報を得る必要がある．

前節のシステムは，差分情報を含むXMLファイルから得た変更箇所の座標を入力することで，編集前後のソースコードの内，該当する箇所の情報を得ることが出来る．しかし，その箇所がソースコード中のどの構文の一部でどのような役割を担っていたかを確認することはできない．そのため，この節では前節のシステムを拡張して編集箇所とその箇所の構文木における親要素から構文情報を取得するシステムを作成する．

木構造に変換したソースコードから構文情報を抽出して木グラフで表示したものを，図6(a)をもとに例として図9に示す．また，構文情報とその概要をまとめた表を表5に示す．



(a) 全体



(b) 赤枠　最上部抜粋

図9 構文情報　木グラフ　例

表 5 タグと内容の対応表

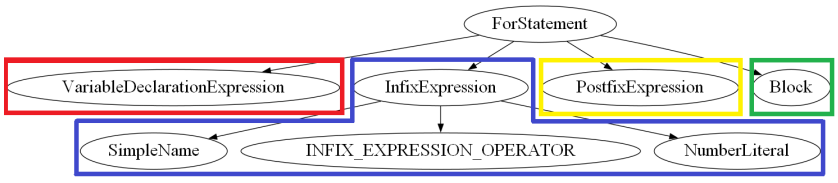
構文情報	概要
ConpulationUnit	抽象構文木のルート
TypeDeclaration	型宣言
Modifier	修飾子
MethodDeclaration	メソッド宣言
SinpleName，QualifiedName	単純名，限定名
block	メソッドやステートメントの範囲
forstatement，Ifstatement	for 文，if 文
VariableDeclarationExpression	変数宣言式
InfixExpression，PostfixExpression	中置演算式，後置演算式
INFIX_EXPRESSION_OPERATOR	中置演算子
NumberLiteral	数値

図 9(b) を確認すると，抽象構文木のルートを表す「ConpulationUnit」が最上部に位置し，そこから子要素としてソースコードの構文情報が連なっている．そのため，今回作成するシステムでは，任意の箇所から type タグを探索し，そこから「ConpulationUnit」を得られるまで直上の親要素の type タグの内容を順に記憶する．

木グラフで表現された構文の例として図 9(a) から青枠で囲われた箇所の for 文を示す「ForStatement」の子要素を抜粋して図 10 示す．

```
1  public class if1 {
2
3      public static void main(String[] args) {
4          for(int i = 0; i < 10; i++){
5              System.out.println(i);
6          }
7      }
8
9  }
```

(a) ソースコード



(b) 構文木

図 10 for 文構文情報　木グラフ　例

図 10(b) の「VariableDeclarationExpression」,「PostfixExpression」,「Block」は本来それぞれに子要素が存在するが, 簡素に表示するために省略している. 図 10 を確認すると, for 文は大まかに 4 つの構文要素で構成されている. また, ソースコードと構文木で対応している箇所を各色の枠で囲って表している. 赤で示された「VariableDeclarationExpression」は変数宣言式, 青で示された「InfixExpression」は中置演算式, 黄色で示された「PostfixExpression」は後置演算式, 緑で示された「Block」は {} 内のループする内容を表す. また,「InfixExpression」はソースコード内における for 文の条件式の内, 終了条件式の「i < 10」に対応する構文情報であり, その子要素はそれぞれ式を構成する各文字を表す. それぞれ, 単純名を表す「SimpleName」は「i」, 中置演算子を表す「INFIX_EXPRESION_OPERATOR」は「<」, 数値定数を表す「NumberLiteral」は「10」に対応している.

変更箇所の親要素の取得の例として, 図 7(b) が示す座標で編集があったものとした場合に, 図 9 の「ConpulationUnit」まで探索し, 記憶される構文情報を図 11 に示す.

実装は 4.2 節のシステムを拡張して行い, 親要素を探索する機能とそれらを配列に格納する機能を追加する. また, label タグに含まれる構文情報に対応する実際ソースコード内の文字もソースコードと構文木との対応を取る上で重要であるため, 同時に変更箇所の label タグの確認を行う. 4.2 節と同様に, ElementTree モジュールを使って xml ファイルの操作を行う. まず, xml 上の変更箇所を特定して label タグと type タグの内容を確認し, その後, 順に親要素を遡って type タグの内容配列に格納する. type タグの内容を配列に格納する際に, その都度内容が「ConpulationUnit」と一致するかの確認を行い, 一致した場合終了する.

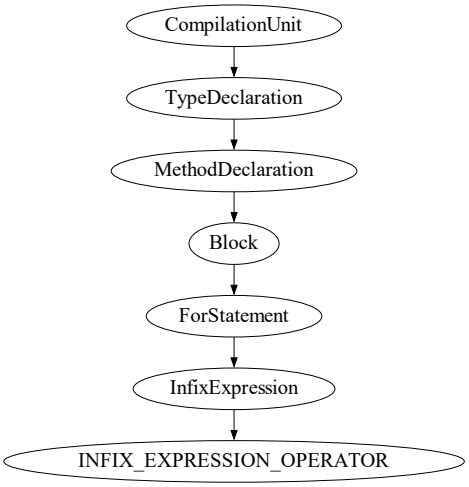


図 11 図 7 変更箇所の親要素取得　例

5　結果・考察

5.1　構文情報とフィードバックへの活用

4章の4.3節，4.4節で作成したツールからフィードバックに有用な情報が得られるか考察する．

4.3節で作成したツールの動作例として，図7(b)の44行目に含まれる座標から変更前のソースコードの該当箇所の構文情報を取得する．入力に図7(b)の44行目に記された座標を与え，図6(b)に示した編集前のソースコードからtypeタグの情報を取得した際の出力結果を図12に示す．結果として，図6のtypeの内容と図12の出力結果が一致していることから，type内の構文情報が取得できていることが分かる．出力されたtypeタグに格納されている内容は入力として与えた箇所の構文情報を表すものであり，ソースコード中の変更箇所の字句の役割を示している．図12で出力されたのは「INFIX_EXPRESSION_OPERATOR」であるため，表5を確認すると中置演算子を表す情報であることが分かる．このことから，変更箇所はある中置演算式の演算子であることが確認できるため，受講者はその箇所を課題中に編集したことが読み取れる．全てのバージョンの差分情報に含まれる各変更箇所の座標をこのツールを用いて特定することによって，課題全体の差分のフローを構成することが出来る．

同様に，4.4節で作成したツールに入力として図7(b)の44行目に記された座標を与え，配列の内容を表示した出力結果を図13に示す．図13に示された出力結果の上に表示された配列がtypeタグの内容，下がlabelタグの内容を表す．typeタグの内容が格納された配列は図11に示した情報と一致しており，システムの出力が正しいことが確認できる．図13を確認すると，取得した要素の中に「ForStatement」,「InfixExpresssion」が含まれていることが確認できる．これらは，変更箇所の「INFIX_EXPRESION_OPERATOR」がfor文とその条件式の中置演算式を構成する要素であることを示している．すなわち，受講者が課題中にfor文の条件式の内の中

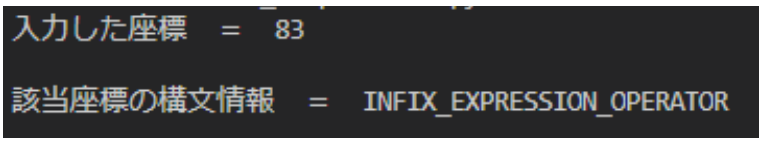


図12該当箇所typeタグ　出力結果

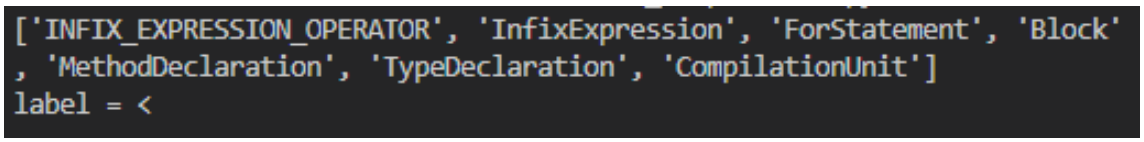


図13図7変更箇所の親要素取得　出力結果

置演算式を編集したことが読み取れる．このように，for文内で編集があった場合には，探索の際に条件式（VariableDeclarationExpression：変数宣言式，InfixExpression：中置演算式，PostfixExpression：後置演算式）とループ内容（block）を表す構文情報のどれか1つを取得するため，条件式とループの内容のどちらで変更があったかを判別することが出来る．このツールを用いて4.3節で作成したツールから得た差分のフローに含まれる各編集箇所の親要素を特定することで，受講者の課題取り組み中のソースコード内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と考えられる．

5.2 提案手法のメリット

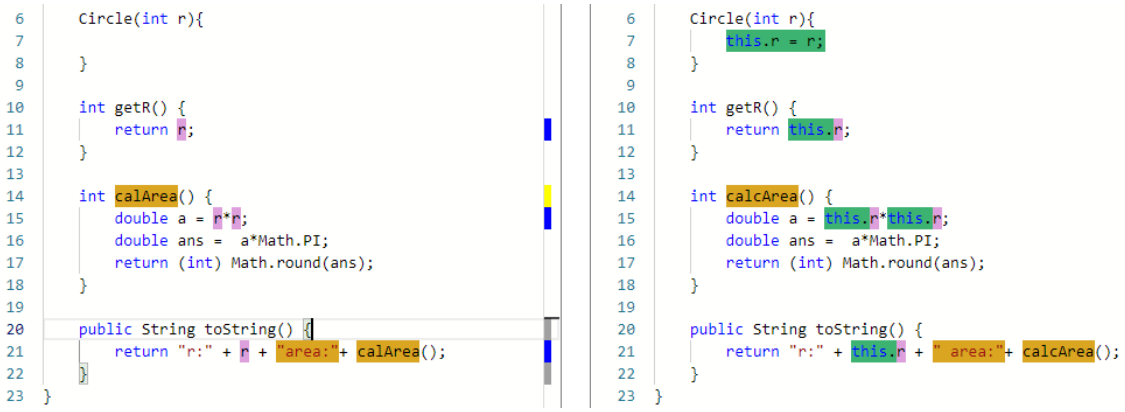
この節では，差分出力手法の特徴や，フィードバックへの活用のメリットを考察する．

5.2.1 差分から得た編集情報のフィードバックへの活用

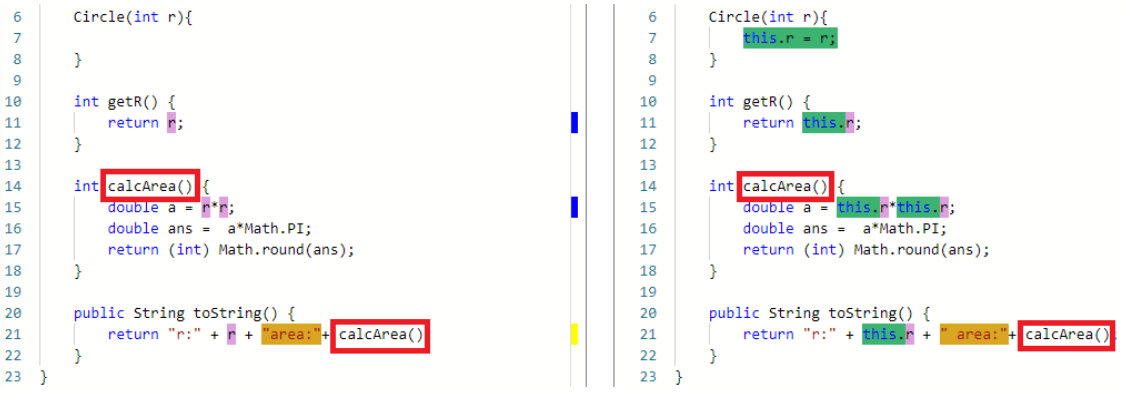
3.1節では，コーディング特徴を流動的に確認し，フィードバックを提示できることを示した．また，図4から課題の取り組みが進むにつれて変更箇所が少なくなっていることが確認できた．各差分出力の際にはscoreが100のソースコードとそれ未満の2つのソースコードを入力しているため，変更箇所として着色されているノードは，各提出においてscoreを100にするために編集が必要な箇所を表していることが分かる．また，これらは各提出において受講者が正しく編集できていない箇所であるため，受講者が課題内で理解が十分でない箇所とも考えることが出来る．これらのことから，課題の進行に沿いつつ，各提出段階で受講者が課題完了に必要な編集箇所を抽出してフィードバックを生成できると考えられる．

例として，円の半径を入力し，その面積を表示するプログラム(Circle.java)を作成する課題にて，4回の提出で課題を完了した受講者が提出したソースコードから出力した差分をブラウザ表示したものと構文木をそれぞれ図14と図15示す．図14, 15はどちらも左側がVer.1〜N-1, 右側がVer.Nのものである．また，赤枠で囲われた箇所は前の提出から更新と挿入の編集が行われた箇所を示す．図14, 15の各提出を確認すると，まず，提出1回目の結果を受けcalAreaメソッドの名前が誤っていることに気づき，提出2回目でcalcAreaに名前を修正していることが読み取れる．提出3回目では，1回目，2回目でthisを用いた変数rの初期化が出来ていなかったことに気づき，修正していることが読み取れる．図15の提出3回目において，右下にある1つのノードが修正できていないことが分かる．この箇所と対応する図14のコードを確認すると，文章を表示するメソッドの内の”area:”であることが確認できる．areaという単語の左に空白が必要なことに気づいておらず，提出1回目から同様の箇所の誤りが続いていることが読み取れる．この受講者は提出

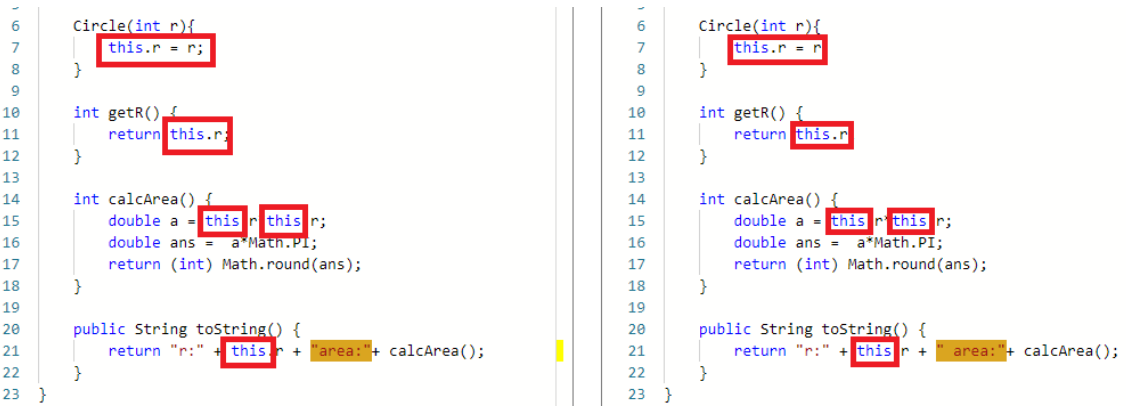
4回目でscoreが100となっているため，最終的にこの誤りを修正して課題を完了したことが分かる．上記のような複数回の提出の間修正されていない誤りが，ある提出で修正されていた場合，受講者が課題取り組み中に誤りの箇所に対する意識が薄くなっており，ふとした時に気づいて修正したことが考えられる．図14，15の例の場合，出力フォーマットの誤りは1回目の提出から確認されるが，3回目の提出まで修正されておらず，4回目の提出で修正されている．同様に，変数rに関する誤りも1回目の提出から確認されるが3回目で修正されるまで複数回にわたり残っている．これらのことから，この情報をフィードバックに用いることで，受講者に課題取り組み中の特定の内容に対する意識の抜け落ちがあることを知るきっかけを与えることが出来る．



(a) 提出 1 回目　結果：Error

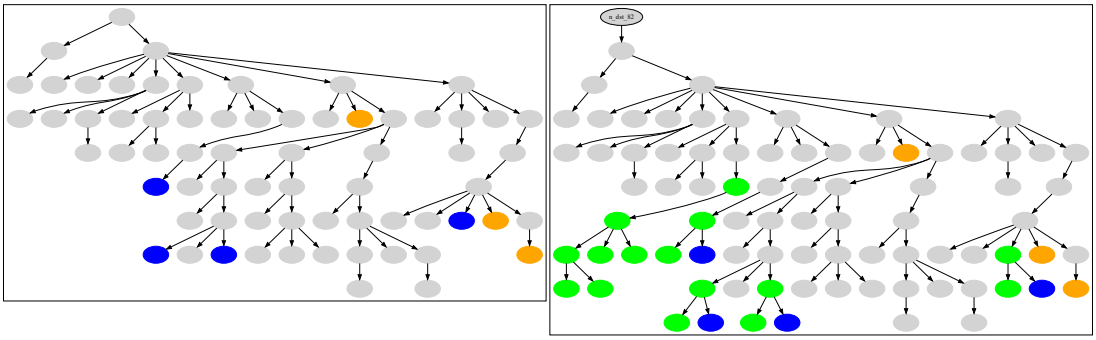


(b) 提出 2 回目　結果：score 0

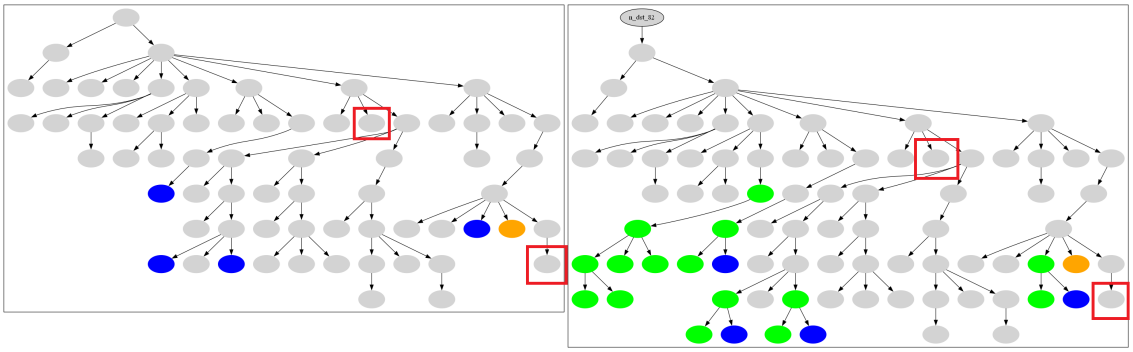


(c) 提出 3 回目　結果：score 75

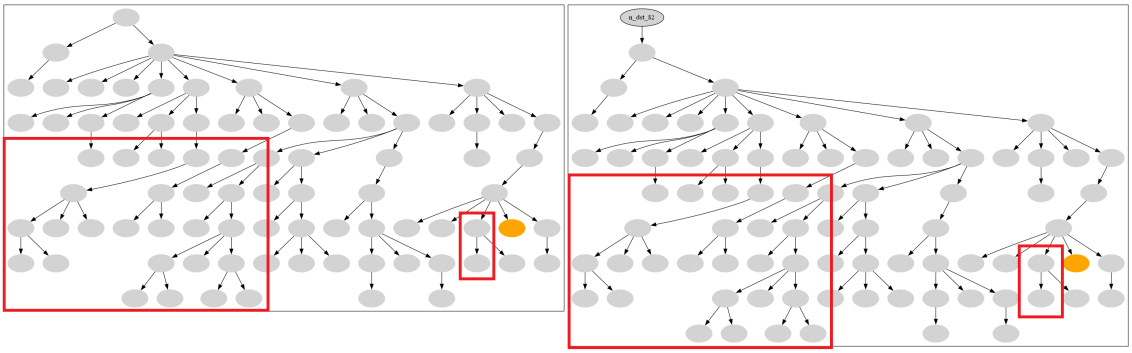
図 14 Circle.java 作成課題差分　ブラウザ表示



(a) 提出1回目　結果：Error



(b) 提出2回目　結果：score 0

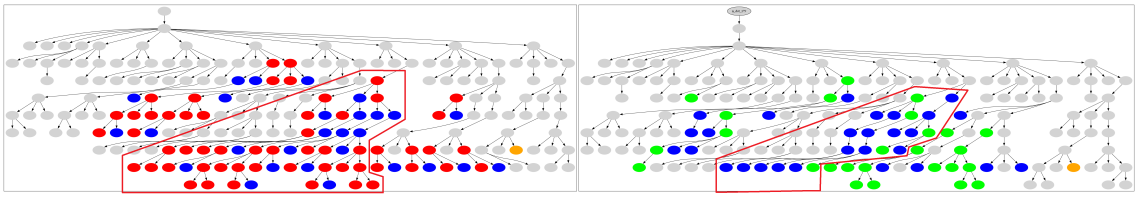


(c) 提出3回目　結果：score 67

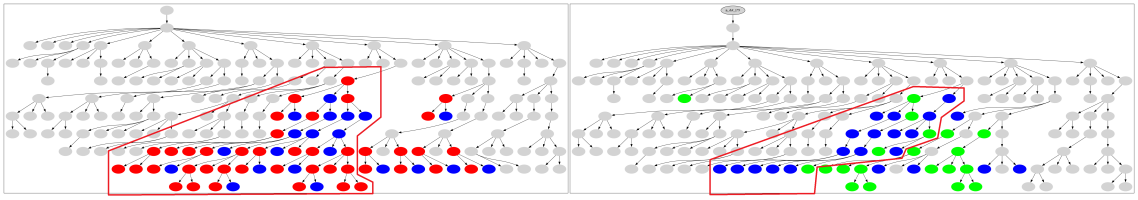
図 15 Circle メソッド作成課題差分　構文木

5.2.2 複雑な課題における有用性

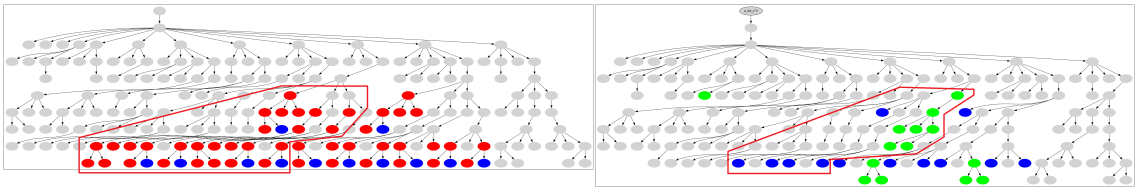
多数のメソッドを含む複雑なソースコードを作成する課題において，提出が多数行われた場合について確認する．その例として，日本円とアメリカドルを表現し，指定した交換レートを基に相互の両替を行うソースコード(Money.java)を作成する課題にて，11回の提出で課題を完了した受講者の提出したソースコードから出力した差分の構文木を図16に示す.



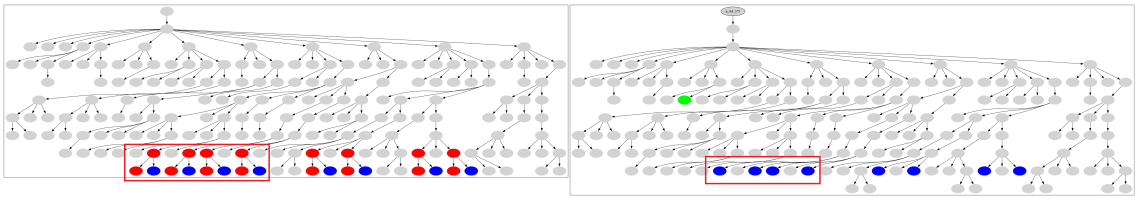
(a) 提出 1 回目　結果：Error



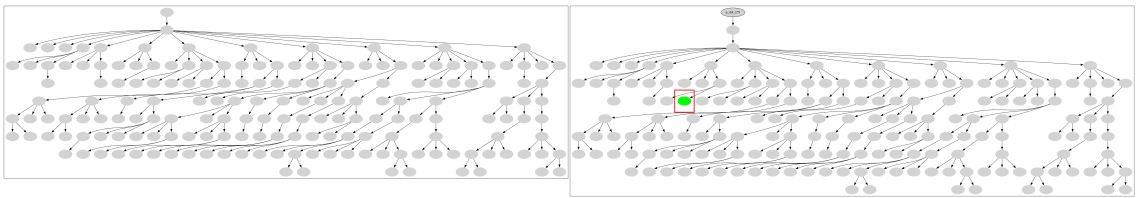
(b) 提出 5 回目　結果：score 50



(c) 提出 7 回目　結果：score 50



(d) 提出 9 回目　結果：score 50



(e) 提出 10 回目　結果：score 50

図 16 Money.java 作成課題差分　構文木　一部抜粋

図16から，課題の取り組みが進むにつれて，図15と同様に変更箇所のノードが減少していく様子が見て取れる．提出10回目の赤枠で囲ったノードは図15の“area:”と同様に，提出1回目から修正されておらず，最終的にこの箇所に単語を挿入する事によって課題が完了している．

提出10回目の該当箇所のソースコードをブラウザ表示したものを図17に示す．図17から，該当箇所の行は交換レートの初期化を行う式であることから受講者は課題を通してこの初期化に気づかなかった事が読み取れ，図15の例と同様のフィードバックが期待できる．

また，提出1回目から9回目まで赤枠で囲った箇所が課題を通して続けて編集されている．各提出での該当箇所のソースコードを図18に示す．図18を確認すると，受講者が続けて編集していたのは指定したレートを基に通貨の両替を行う“exchangeメソッド”であることが分かる．提出1回目，2回目では，メソッド内全体が編集されており，特定の単語の誤りを確認しづらい．しかし，課題の取り組みが進むにつれてメソッド内の編集が減少し，提出9回目にはif文内の計算式の変数が誤っていることが特定可能になった．受講者は提出9回目で変数amountと変数rateに“this”が不必要な事に気づき，修正する事でその後の提出で該当箇所を解決したと読み取れる．受講者が課題開始から提出9回目まで“this”に関する編集を繰り返し，誤りを続けているという情報から，受講者のこの情報をフィードバックに用いることで，受講者自身に“this”を用いたローカル変数とフィールド変数の区別に対する理解が不十分であることを気づかせることが出来ると考えられる．

これらのことから，多数のメソッドを含む複雑なソースコードにおいて，初回の提出で誤りの箇所が多く見られる場合でも，最終提出までの差分を確認し，明確な誤りの情報を抽出することで，有効なフィードバックを生成することが出来ると考えられる．

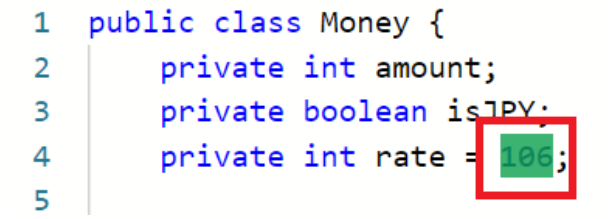
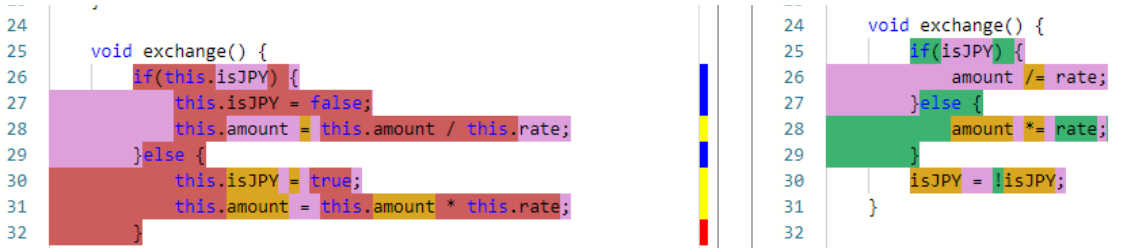


図17 提出10回目　赤枠箇所



(a) 提出 1 回目

(b) 提出 5 回目

(c) 提出 7 回目

(d) 提出 9 回目

図 18 図 16 提出 1〜9 回目　該当箇所

5.2.3 前後の提出間での差分出力手法との差異

オープンソースプロジェクトを対象とした分析において，ソースコード間の差分を用いる際には連続する2バージョン間の編集間の差分を出力する．今回の環境でその差分出力手法を用いた場合と，本研究の提案手法との差異の考察を行う．

例として5.2.1で示したCircle.javaを作成する課題にて提出された4つのソースコードから，Ver.1と2，Ver.2と3，Ver.3と4で差分を出力し，ブラウザ表示したものを図19に示す．図19と図14を比較する．図19では，各提出間の編集の動きが明確に表れており，受講者がどの提出のタイミングでどのような編集を行ったかという情報のみを抽出し易い．しかし，5.2.1項と5.2.2項で説明したような，課題取り組み開始から終了までの一連の提出から得られる編集箇所の増減の流れを表す情報は得られない．そのため，受講者の意識の抜け落ちや，継続した誤りが読み取れるようなフィードバックに有効な情報が得られない．また，前後の差分を出力する場合には，比較対象が必ずしも正しい仕様のソースコードではないため，編集箇所が正しい編集を行っていると断言することは出来ない．その点，本研究の提案手法では，比較対象が正しく作成されたソースコードであるため，各編集箇所を受講者の誤りの箇所として見ることが出来る．また，あらかじめ用意した模範解答ではなく，受講者自身のscore100のソースコードを比較に用いることによって，各受講者のコーディング特徴に合わせて分析を行うことが出来る．以上の事から，提案手法は前後の提出間で差分を出力する手法よりも受講者の提出したソースコードからフィードバックに必要な情報を抽出するのにに適していると考えられる．

```java
public class Circle {
    private int r;

    Circle(int r){

    }

    int getR() {
        return r;
    }

    int calArea() {
        double a = r*r;
        double ans =  a*Math.PI;
        return (int) Math.round(ans);
    }

    public String toString() {
        return "r:" + r + "area:"+ calArea();
    }
}
```

```java
public class Circle {
    private int r;

    Circle(int r){

    }

    int getR() {
        return r;
    }

    int calcArea() {
        double a = r*r;
        double ans =  a*Math.PI;
        return (int) Math.round(ans);
    }

    public String toString() {
        return "r:" + r + "area:"+ calcArea();
    }
}
```

(a) Ver.1 Ver.2 間

```java
public class Circle {
    private int r;

    Circle(int r){

    }

    int getR() {
        return r;
    }

    int calcArea() {
        double a = r*r;
        double ans =  a*Math.PI;
        return (int) Math.round(ans);
    }

    public String toString() {
        return "r:" + r + "area:"+ calcArea();
    }
}
```

```java
public class Circle {
    private int r;

    Circle(int r){
        this.r = r;
    }

    int getR() {
        return this.r;
    }

    int calcArea() {
        double a = this.r*this.r;
        double ans =  a*Math.PI;
        return (int) Math.round(ans);
    }

    public String toString() {
        return "r:" + this.r + "area:"+ calcArea();
    }
}
```

(b) Ver.2 Ver.3 間

```java
public class Circle {
    private int r;

    Circle(int r){
        this.r = r;
    }

    int getR() {
        return this.r;
    }

    int calcArea() {
        double a = this.r*this.r;
        double ans =  a*Math.PI;
        return (int) Math.round(ans);
    }

    public String toString() {
        return "r:" + this.r + "area:"+ calcArea();
    }
}
```

```java
public class Circle {
    private int r;

    Circle(int r){
        this.r = r;
    }

    int getR() {
        return this.r;
    }

    int calcArea() {
        double a = this.r*this.r;
        double ans =  a*Math.PI;
        return (int) Math.round(ans);
    }

    public String toString() {
        return "r:" + this.r + " area:"+ calcArea();
    }
}
```

(c) Ver.3 Ver.4 間

図 19 Circle.java 作成課題前後差分　ブラウザ表示

5.3 今後の拡張性

提案手法の実装とそこから得た結果を基にした考察から，提案手法の拡張性について考察する．

5.3.1 差分間の編集行動

提案手法では，Gumtreeで差分を出力した際に検出する編集行動を基にフィードバックを生成する．ソースコード間での編集行動の内,「挿入」と「更新」と「削除」は編集前後の単語や句の変化した時に検出されるのに対し,「移動」は変化がない単語や句が異なる座標に移動した時に検出される．フィードバックを生成する際は，各構文に含まれる単語や句の変更を基に受講者のコーディング特徴を確認するため，変更の無い単語や句を検出してしまう「移動」は今回の提案手法ではノイズとなる．4章や，5.2.1項で示した差分の構文木やブラウザ表示では移動（構文木では紫色，ブラウザ表示では青色）と削除（赤色）は差分情報の内の高い割合を占めていることが確認できる．そのため,「移動」をフィードバックに用いると他の有効な情報を受講者に知らせることを阻害する可能性がある．このことから，今後フィードバックシステムを作成する際には,「移動」情報を省いた他の編集行動の情報で構成することで，よりフィードバック情報の生成に有用な出力が作成できると考えられる．

5.3.2 該当箇所とその親要素以外の構文情報

4.3節では該当箇所の親要素を探索するツールを作成した．このツールは該当箇所から順に直上の親要素を辿っていくため，得られる構文情報は該当箇所とその親要素に含まれるもののみである．そのため，該当箇所の親と同じ親を持つ子要素の情報は得ることが出来ない．図6に示したような演算子が同一の親をもつ子要素には4.3節の図10(b)のようにその演算子を用いて演算を行う変数や数値の情報が含まれている．ソースコードの数式や条件式などを分析する場合，式を構成する各文字だけでなく，式全体の情報を得ることで，変数が何に初期化されているか，どのような演算かを確認することが出来る．しかし，Gumtreeによる差分では4.2節の図6のように式を構成する字句単位でしか情報を得ることが出来ない．

これらのことから，今後の拡張性として該当箇所の親要素を探索すると同時に該当箇所と同じ親を持つ要素を探索し，その構文情報を得るツールを作成することが考えられる．式を構成する字句に関連性を持たせて，それを基にフィードバックを生成できれば，条件式や演算式において誤りがあった場合に式の構成とその演算結果の関係の理解が不十分であることを示すフィードバックの生成が可能になると考えられる．

6 おわりに

本研究では，授業での1人の受講者が取り組んだある1つ課題において，最終版のソースコードとそれ以前に提出された複数のソースコードから構文木を出力し，それらの連続した差分を出力することで正解となるソースコードを作成するために必要な変更箇所を出力する手法を提案した．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差分フローを求め，構文木情報を付与することで，それらの親要素から機械的処理に必要な構文情報や，ソースコード内の各構文に対する編集行動を示す構文情報を得られる．

今回の提案手法の内，差分出力手法ではGumtreeを用いて最終バージョンとそれ以前の各バージョン間の差分を出力する．これによって，課題取り組み開始から完了までの差分の増減を構文情報を付与して取得することが出来る．このメリットとして以下のことが考えられる．

- 差分箇所の増減から，受講者の課題中の理解不十分箇所や特定の内容の意識の抜け落ちを読み取れる
- 誤りの箇所が多い複雑なコードでも受講者の誤りの特徴が明確になる
- 比較対象が受講者が作成した正答のため，編集箇所を誤りの箇所と見ることが出来る

今回作成したツールは本研究の提案手法の内，差分フローに含まれる各変更箇所の座標から変更箇所を特定するものとそれらの親要素を取得するものである．作成したツールにある学生が提出した課題を入力として与えた際の出力結果から，変更箇所がどの構文に含まれるかを判別することが出来る構文情報が得られた．作成したツールを用いて差分フローに親要素の情報を付与することで，受講者の課題取り組み中のソースコード内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と考えられる．

本研究の発展として以下の要素が挙げられる．

- 「移動」情報によるノイズを削除，フィードバック作成する際に「移動」情報を省いた編集行動の情報で構成する
- 編集箇所と同じ親を持つ構文情報を取得することで式を構成する字句に関連性を持たせる

7　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, 常日頃から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.

8 参考文献

参考文献

[1] 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.

[2] Hui Sun, Bofang Li, Min Jiao:”YOJ: An online judge system designed for programming courses”, 9th International Conference on Computer Science Education (ICCSE2014), pp.812-816, 2014.

[3] Wenju Zhou, Yigong Pan, Yinghua Zhou, Guangzhong Sun. The framework of a new online judge system for programming education. Proceedings of ACM Turing Celebration Conference (TURC2018), pp.9-14, 2018.

[4] 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.

[5] 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.

[6] 大谷明央, 肥後芳樹, 楠本真二, 編集スクリプトへのコピーアンドペースト操作の導入によるコード差分の理解向上の試み, 情報処理学会論文誌, Vol.58, No.4, pp.833-844, 2017

[7] 西城戸星龍, プログラミング授業のソースコード提出履歴を用いた理解の有無の予測, 奈良工業高等専門学校情報工学科令和3年度卒業研究論文, p.9, 2022.