卒　業　研　究　報　告　書

令和7年度

研究題目

提出履歴のAST差分情報に基づく
構文理解度の予測手法

指導教員　　上　野　秀　剛　　准教授

氏　　名　　小　林　瞳　子

令和08年01月27日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

提出履歴のAST差分情報に基づく構文理解度の予測手法

上野研究室　小林瞳子

プログラミング講義では，課題やテスト結果に基づく総合評価が一般的であり，条件分岐やクラスなどの構文要素ごとの理解度を詳細に把握することは困難である．本校で用いられているOnline Judge System(OJS)は提出プログラムを自動採点し履歴を蓄積するが，プログラム構造や使用構文は評価対象ではない．一方，提出履歴から抽象構文木(AST)に基づく差分フローを抽出することで，完答に至るまでの修正過程を構文要素単位で捉えられる．そこで本研究ではOJSの提出履歴から得られる差分フローを用い，クラスに関する構文要素の理解度を定量的に算出する手法を提案する．対象とする構文要素は，インスタンス生成，this，アクセス修飾子，メソッド呼び出し，戻り値の型，メソッド引数の6要素であり，これらを3つの構文要素グループに分類した．完答に至るまでに発生した各構文要素の差分数を誤用数として算出し，最終提出時の構文要素利用数に対する割合を正規化することで，提出回数やコード規模に依存しない理解度指標を定義した．提案メトリクスの有用性を検証するため，算出した理解度と受講者による主観評価との相関分析を行った．講義期間中に3回実施したアンケートと，対応する課題回の提出履歴を用いて分析を行い，複数の主観評価項目を用いる場合にはCronbach's αにより内的一貫性を確認した．分析の結果，講義後半のアンケートにおいてメソッド・コンストラクタに関する構文の理解度と，関連する主観評価との間に有意な正の相関が確認された．また，クラスに関する全構文要素の理解度についても，複数の主観評価項目を統合した場合に有意な相関が得られた．一方，単一項目による主観評価との間には有意な相関が見られない場合もあった．以上より，複数の関連する構文要素や評価項目を統合することで，受講者の理解度をより安定して捉えられることが示された．本手法は構文要素単位で理解度を自動算出でき，受講者への学習フィードバックや教員による指導支援への活用が期待される．今後は，if文やfor文など他の構文要素へ適用範囲を拡大することが課題である．

目 次

1 はじめに

高専や大学のプログラミング講義では，受講者の理解度を課題やテストの結果に基づいて総合的な評価を行い，個々の構文要素について評価を行わないことが一般的である．そのため，受講者が課題に取り組む過程でどの構文要素につまずいているのか，あるいはどの構文要素は理解して使用しているのか，といった学習過程での情報が評価に反映されない．よって，各構文要素に対して受講者がどの程度理解しているのか詳細に把握することが困難であるという問題がある．

プログラミング講義における評価方法の一つとして，本校のプログラミング講義ではOnline Judge System(OJS)が活用されている．OJSは受講者が提出したプログラミング課題のソースコードに対して，自動的にコンパイル・実行およびテストケースによる採点を行い，結果を即時にフィードバックするシステムである．受講者はフィードバックをもとに自身のソースコードを修正し再提出を繰り返すことで，最終的に100点を取ることを目標とする．この再提出の繰り返しによって，OJSには受講者が課題を解く際の誤答から完答への一連の過程が提出履歴として蓄積される．しかし，採点にはテストケースに対する実行結果のみを参照しており，提出されたプログラムの中でどのような構文が用いられているか，どのような構造になっているのかなどは評価対象ではない．つまり，個々の構文要素に対する理解度については評価できていない．

個々の構文要素に対する受講者の理解や正誤を評価することを目的に，OJSの提出履歴から，受講者が100点を取ったソースコードとそれ以前のソースコード間の差分構文木を自動的に抽出するツールが提案されている.[1]. このツールは提出されたソースコード群を抽象構文木(AST: Abstract Syntax Tree)で表現し，バージョン間の差分を時系列に並べた「差分フロー」を生成する．差分フローは受講者が完答した最終版とそれ以前のソースコードの差分群であり，完答に至るまでの修正内容が構文要素単位で確認できる．そのため，ある構文要素に対する修正履歴が多く存在する差分フローは，提出者がその構文要素に対する理解が不十分である事を示す.

本研究は構文要素単位での受講者の理解度を定量的に計測することを目的に，ソースコードの提出履歴から抽出した差分フローから理解度を算出する手法を提案する．提案手法を評価するために，本研究ではコンストラクタやクラス内メソッドといった，クラスに関する構文を対象に評価実験を行う. クラスの概念はオブジェクト指向プログラミングにおける重要な概念であり，インスタンス生成，メソッド呼び出し，アクセス修飾子など複数の構文要素を組み合わせて利用する．これらの構文要素は相互に関係しており，いずれか一つの理解が不十分であっても，クラスを意図した通りに利用することは困難である．そのため，クラスに関する理解度の違いは，クラスを構成する個々の構文要素に対する理解不足，ある

いはそれらの構文要素同士の関係性に対する理解不足として表れると考えられる．本研究では本校のプログラミング講義で用いられているOJSに蓄積されたデータを利用した評価実験を行う．対象講義はJava言語を用いて，オブジェクト指向やクラスの扱い方を中心に学習し，講義内容に対応する課題が複数課されている．そのため，課題の提出履歴から得られる差分フローにはクラスに関する構文が多く含まれており，クラスの設計や利用に関する受講者の理解状況を分析する対象として適している．分析では提案手法によって算出する理解度と，クラスに関する構文の理解度を問う主観評価アンケート結果の相関を分析する．提案手法によってクラスに関する構文要素の理解度を測ることが可能になれば，受講者それぞれの理解状況に合わせた学習支援の実現を期待できる．

2 関連研究

2.1 スキルツリーによるプログラミングスキル表現手法[2]

プログラミング講義の受講生の各構文要素に対する理解度を把握するために，プログラミングスキルを構文要素に細分化し，それらの関係をスキルツリーとして表現することで，知識の構造を可視化する手法を提案した研究がある[2]．このスキルツリーを受講者が用いることで，受講者は自身の構文要素の習得状況を視覚的に把握できる．教員が用いる場合，各受講者の理解状況を把握することで，適切なフィードバックや指導支援につなげられる．これは，従来の課題やテストによる総合評価では把握が困難であった，構文要素単位での理解状況を分析可能にする点で有用である．

本研究はスキルツリーによって可視化された各構文要素に対して理解度を定量化するメトリクスを提案することで，各構文に対する理解状況をより具体的に把握し，理解を促進するためのフィードバック生成につなげることができる．

2.2 ソースコードの提出履歴を用いた研究

プログラミング教育において自動採点システムやオンラインでの演習環境の普及に伴い，受講者が提出したソースコードの履歴を分析対象とする研究がおこなわれている．三野はOJSを用いた講義を対象として，誤答の原因を十分に考察せずに修正を繰り返す無作為修正者の連続した提出の有無や提出回数と言った提出行動の特徴と，課題の行き詰まりとの関係を分析している[3]．西城戸は理解が不十分なため完答までに時間がかかる受講者の予測を目的に，受講者が提出したソースコードのスナップショットから完答までにかかる時間を予測する手法を提案している[4]．これらの研究は提出回数や提出にかかった時間といった提出行動に基づいて受講者の状態識別をしている．これに対して本研究ではASTで表現されたソースコード差分からソースコード中の構文要素の利用状況を抽出し，その差分数から受講者の理解度を定量的に評価する点に特徴がある．

青木はOJSに蓄積された受講者のソースコードの提出履歴から，最終提出のソースコードとそれ以前に提出されたソースコード間の差分のASTを抽出し差分フローを生成する手法を提案している[1]．本研究ではこの手法を利用して差分フローからクラスに関する構文の差分のみを抽出し，その差分数を基に理解度を算出する．

2.3 ASTを用いた研究

ソースコードの差分を扱う研究において，プログラムの構文構造を木構造で表現できるASTを用いた手法が複数提案されている．Malyshevaらは学生が提出し

た誤りを含むソースコードに対して修正案を提案し，その修正がコードの挙動をどのように変えるか示すアルゴリズムを提案している[5]. この研究では誤ったソースコードと他学生の正解コードとの間の編集スクリプトの導出にAST表現を用いている. 鄭らは対象のソースコードと複数の正解パターンとの類似度をASTや変数値の系列の比較によって求める手法を提案している[6]. 涌井らは対象ソースコードと正解ソースコードのASTを生成し，Path Contextと呼ばれるデータ表現を用いて類似度を算出し，プログラム部分点を算出する手法を提案している[7]. これらの研究はすでに用意されている正解ソースコードと対象ソースコードとのASTの類似度を算出している. 本研究では正解ソースコードではなく，受講者自身の最終提出ソースコードとそれ以前の提出ソースコードとの間でASTの差分の数を求め，理解度を定量的に示すことを目的としている.

漆原らはASTを用いてプログラミング理解度を算出する手法を提案している. この手法は課題文が意図する正解のASTを生成し，そのASTと受講者の提出したプログラムのASTから理解度を算出している[8]. 本研究は受講者の提出履歴から生成したASTの差分フローからクラスに関する構文の差分数を数えて理解度の算出を試みている点が異なる.

2.4 本研究の位置づけ

本研究はプログラミング教育における受講者の理解度を構文要素単位で定量的に示すことを目的としている. 本提案手法はOJSに蓄積された受講者のソースコードの提出履歴からAST表現を利用した差分を抽出し，その差分数から理解度を算出する. ソースコードの提出履歴を用いた従来研究では，提出回数や提出に要した時間，連続提出の有無といった提出行動の特徴から受講者の状態を識別する手法を提案している. これらの手法は課題に行き詰まっている受講者の早期発見に有効である一方で，ソースコード中のどの構文要素がどの程度理解されているかといった，構文単位での理解状況を定量的に示していない. また，ASTを用いた研究では正解ソースコードと対象ソースコードとの類似度算出や編集操作の抽出を通じて，プログラムの構文構造に着目した評価やフィードバック生成が行われてきた. しかし，これらの研究の多くはあらかじめ用意された正解プログラムとの比較に基づくものであり，学習者自身の試行錯誤の過程を考慮した理解度評価には至っていない.

本研究では受講者のある課題に対する最終提出のプログラムを基準とし，それ以前の提出との間で生成した差分フローに着目する. 差分フローからクラスに関する構文要素の差分のみを抽出し，その差分数を基に理解度を算出することで，学習者が最終的に到達した解に至るまでの過程を反映した構文単位での理解度評価を試みる点に特徴がある.

3 準備

3.1 Online Judge System

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・実行する．実行時にあらかじめ用意された入力を与え，その出力が，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は用意されたテストケースに対して正解した数から点数*score*を計算する．

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

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を完答したと見なし，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*とともに提出履歴として記録される．

提出履歴には受講者が課題を完答するまでに行った編集過程が時系列で保存されている．最終提出のソースコードとそれまでの提出ソースコードとの間の差分は，完答に至るまでに各構文要素に対する修正の様子が含まれている．そのため，ある構文要素に対して修正数が多ければ理解不足，修正数が少なければ理解している，と推定できる．

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)ソースコードと，(b)対応するASTを示す．(b)の各ノードはプログラム構文のある要素で，構文情報，対応する文字列をラベルとして持つ．表1に構文情報の一部を示す．ASTは順序木であり，親ノードと枝で結ばれた子ノードは親についての詳細情報を表す．例えば，図1(b)の下線部のノード“NumberLiteral:1”は，図1(a)のソースコードの3行目のi==1の1に対応する要素であり，構文情報NumberLiteralは数値定数を，文字列1が値を示す．また，図1(b)の破線で示したノードはif文を意味しており，図1(a)の破線で囲んだif文全体と対応している．子ノードはif文を構成する2要素である1)条件式(InfixExpression)を表すノードと，2)return文(ReturnStatement)を表すノードを表す．

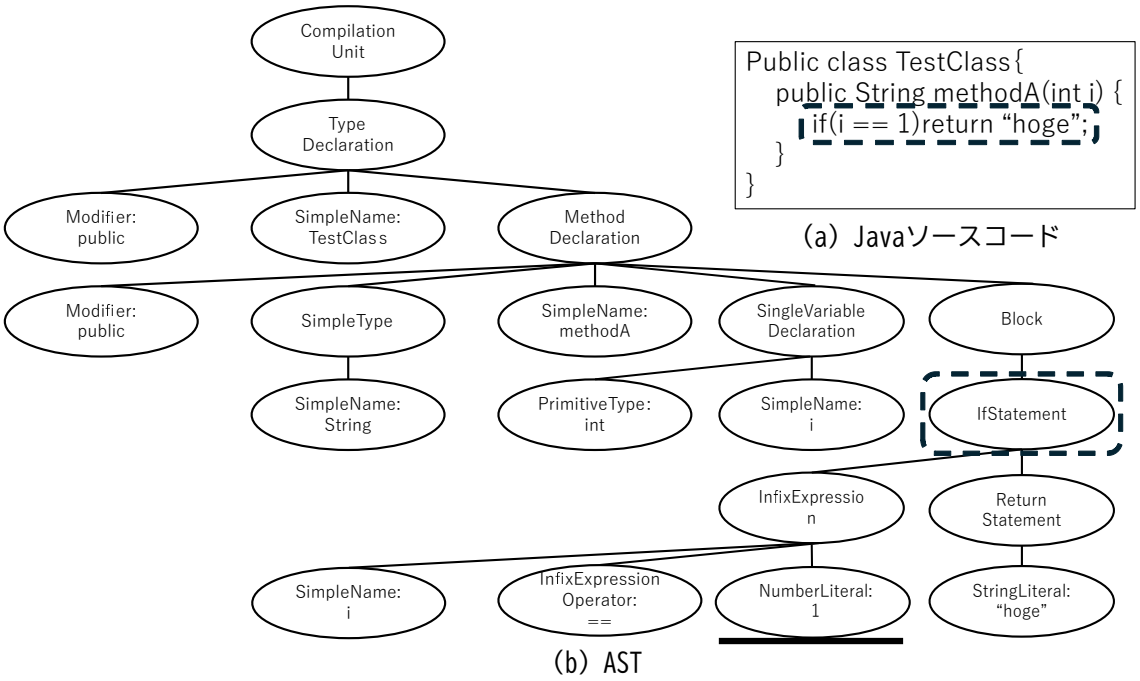

図 1 ソースコードと AST

AST は構文要素単位でプログラムを表現するため，AST の差分を用いることで構文要素単位の差分を得ることができる．本研究では AST の差分情報を元に構文要素レベルでの詳細な評価を行う．

3.3 ソースコード間の差分フロー [1]

図 2 に本研究で用いる差分フロー生成システムの流れを示す．まず，1) OJS に蓄積された各受講者の提出履歴から，同一課題に対して提出された複数のソースコードを時系列に取得する．次に，2) 差分解析器 GumTree を用いて最終提出されたソースコードとそれ以前の各提出との間でコード差分を抽出する．抽出されたコード差分は，文字列レベルでの変更を表すものであり，どの構文要素に対する編集であるかは分からない．そこで差分フロー生成システムでは，3) 各ソースコードを AST に変換する．その後，4) 各ソースコードの AST とコード差分を差分フロー抽出ツールに入力し，AST で表現された差分を算出する．AST 差分では，追加・削除・更新といった編集操作が，条件分岐，繰返し，メソッド，クラスなどの構文要素単位で表現される．例えば，for 文の条件式やクラス内メソッドの定義に対する変更は，それぞれ対応する構文ノードに対する編集として抽出される．AST 差分を用いることで，受講者が課題に取り組む過程において，どの構文要素に対してどの程度の修正を行っているかを把握できる．特定の構文要素に対して編集が繰り返されている場合，その構文に関する理解が十分でない可能性がある一方で，少ない編集で正しく利用されている構文要素は，理解に基づいて使用され

表 1 構文情報

構文情報	概要
CompilationUnit	対象ソースコード全体
TypeDeclaration	型宣言
Modifier	修飾子
MethodDeclaration	method 宣言
MethodInvocation	メソッド呼び出し
METHOD_INVOCATION_RECEIVER	メソッドを呼び出す対象 (レシーバ)
METHOD_INVOCATION_ARGUMENTS	メソッド呼び出しの引数
SimpleName	単純名
Block	method や Statement の範囲
ExpressionStatement	式文
IfStatement	if 文
VariableDeclarationExpression	変数宣言式
InfixExpression	中置演算式
NumberLiteral	数値定数
StringLiteral	文字列リテラル
ClassInstanceCreation	クラスのインスタンス生成
ThisExpression	this
SingleVariableDeclaration	引数
SinpleType	クラス型・インタフェース型
PrimitiveType	基本データ型

ていると考えられる.

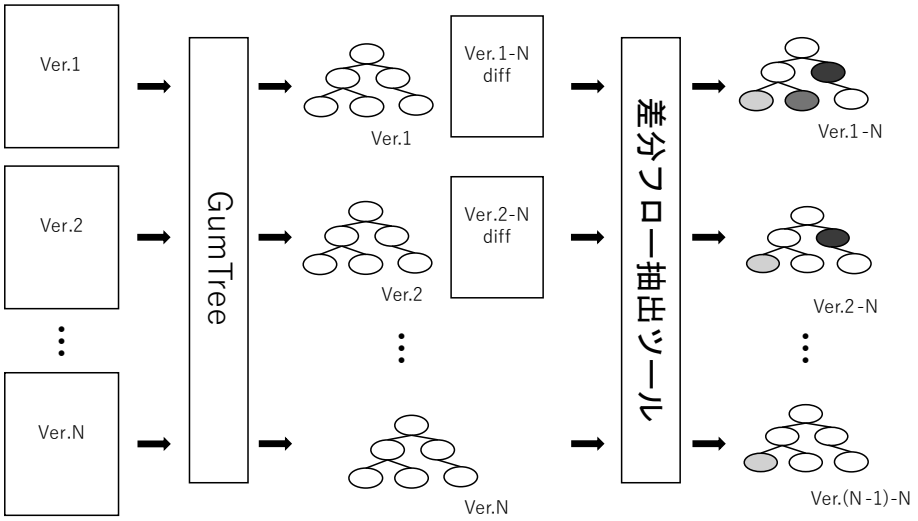

図2差分フロー抽出システム

4 提案手法

4.1 対象とする構文要素

本研究ではクラスに関する構文の理解度を示すメトリクスを提案する．クラスはオブジェクト指向プログラミングの重要な概念である．データと処理を一体として設計・利用するための基本的な抽象化機構であり，適切に用いることができているかは，オブジェクト指向プログラミングに対する理解度を反映すると考えられる．一方で，クラスの設計・利用は初学者にとって理解が難しく，構文的には正しいコードであっても，他の構文要素とのつながりや似た概念の区別などが理解できていないような誤りがよく見られる．そのため，クラスに関する構文が含まれるソースコードの履歴を分析することで，受講者のオブジェクト指向プログラミングに対する理解状況を推測できると考えられる．本稿の提案手法ではJava言語を対象に，クラスに対する理解の有無を反映する，以下の6つの構文要素を抽出する．

- インスタンス生成
  定義されたクラスを基に，プログラム実行時にメモリ上へ具体的なオブジェクトを生成する処理である．例えば，Java言語においてStudentクラスが定義されているとき，次のコードはStudentクラスのインスタンスを生成する．

```
Student s = new Student("Kobayashi");
```

  このとき，new Student("Kobayashi")により，Studentクラスのインスタンスが生成され，生成されたオブジェクトへの参照が変数sに代入される．これにより，sを通じてこのインスタンスが保持する属性やメソッドを利用すること

が可能になる．インスタンス生成を正しく記述するためには，クラスとインスタンスの関係を理解している必要がある．インスタンスを生成せずにクラスから直接インスタンスメソッドを呼び出そうとする記述や，未定義のコンストラクタを呼び出そうとする記述は，インスタンス生成を理解していないことを表す．

- this
  クラス内メソッドなどで利用される，自分自身のインスタンスを指す参照である．以下のStudentクラスにフィールドnameが定義されており，コンストラクタ内でも同じnameという変数が仮引数として宣言されてる例を考える．このときthis.nameはフィールドのnameへの参照を表し，仮引数のnameと区別することができる．

```
class Student {
    String name;

    Student(String name) {
        this.name = name;
    }
}
```

  this参照を正しく記述するためには，フィールドと引数の違いや，thisがインスタンスの状態を参照する文脈であることを理解している必要がある．例えば，コンストラクタにおいてフィールドと同名の引数が存在するにもかかわらず，thisを用いずに代入を行う記述は，意図したインスタンス変数への代入が行われないため，thisを理解していないことを表す．

- アクセス修飾子
  クラスやフィールド変数の利用範囲を指定する概念である．例えば，publicはすべてのクラスからアクセス可能であり，privateは同一クラス内からのみアクセス可能である．アクセス修飾子を正しく記述するためには，どの修飾子がどのような効果を持つか理解している必要がある．例えば，フィールドやメソッドで無差別にpublicを宣言している場合や，本来インスタンスに対して呼び出されるようなstaticを付与すべきでないメソッドに誤って付与してしまっている場合などは，アクセス修飾子を理解していないことを表す．

- メソッド呼び出し
  プログラムで定義した関数や標準ライブラリのメソッドに処理を依頼し，実行させることである．例えば，次のコードはStudentクラスのインスタンスを生成した後，Studentクラスで定義されているnumberメソッドを呼び出している．

```
Student s = new Student("Kobayashi");
s.number(20);
```

このとき, s.number(20)の部分がメソッド呼び出しである. メソッド呼び出しについて正しく記述するためには, インスタンスメソッドが生成されたインスタンスを対象として処理されることや, 呼び出し元となるインスタンスを用意する必要があることを理解している必要がある. インスタンスメソッドをクラス名から直接呼び出そうとする記述や, どのオブジェクトに対して処理が実行されるか不明確な記述が見られるとき, メソッド呼び出しを理解していないことを表す.

- メソッド戻り値の型設定

  次のようなメソッドが定義されている. このとき, 1行目int addのintはこのメソッドの戻り値の型を指定している. これをメソッド戻り値の型設定とする.

```
private int add(int a, int b) {
        return a + b;
    }
```

メソッド戻り値の型設定を正しく記述するためには, メソッドが呼び出し元へ返す値の型をあらかじめ宣言する必要があることを理解している必要がある. また, メソッドとコンストラクタの違いについても理解している必要がある. 正しく理解していない場合, return文で返す値の型と一致しなかったり, 値を返しているにもかかわらずvoidを指定する記述が見られる. また, コンストラクタでは戻り値の型を設定してはいけないが, 誤って指定していることでメソッドとして宣言されているような記述もメソッドとコンストラクタの違いを理解していないことを表す.

- メソッドの引数

  次のようなメソッドが定義されているとき, int aやint bと書かれている部分がある. このメソッド宣言の引数部分をメソッドの引数部とする.

```
private int add(int a, int b) {
        return a + b;
    }
```

メソッドの引数を正しく記述するためには, 引数は型と名前の組で宣言することや, メソッド呼び出し部の引数と順序や型が一致する必要があることを理解しなければいけない. 正しく理解していない場合, 型や名前が抜けている記述や, クラスであればフィールドと混合している記述, 呼び出し部と順序や変数の数が一致しないような記述がみられたとき, 理解していないことを表す.

本研究では6つの構文要素を以下の3つのグループに分類し，各分類に対する差分の数を分析に用いる．

- MTCON: メソッド・コンストラクタ
  - メソッド呼び出し
  - メソッド戻り値
  - メソッド引数
- CLASS: クラス定義・利用
  - インスタンス生成
  - this
  - アクセス修飾子
- ALL: すべて
  - 6つの構文要素すべて

MTCONはメソッド・コンストラクタに関する構文要素グループで「クラスに属する処理の定義」という共通した役割を持つ．メソッドやコンストラクタの利用においては，どのように呼び出され，どの値を受け取り，どの値を返すかというインタフェース部分の理解が重要になる．CLASSは，クラスの定義・利用に関する構文要素グループで，オブジェクトを正しく利用する際に不可欠な要素である．ALLは6つの構文要素すべてを含むグループで，広くクラスに関する構文を包括的に捉えるため設定する．

4.2 提案メトリクス

各構文に対する理解度を示すメトリクスを提案する．ある受講者がある課題に対して完答する（*score*が100になる）までに提出したソースコード群において，クラスに関する構文を利用した回数と，そのうち誤用した回数から理解度を表現する．理解度は以下の式4.2.1で算出する．

$$\text{理解度} = (1 - \frac{\text{誤用数}}{\text{延べ利用数}}) \times 100 \tag{4.2.1}$$

誤用数は完答したソースコードとそれ以前に提出したすべてのソースコード間の，クラスに関する構文の差分数である．完答のソースコードと，それ以前のソースコード間で検出された差分を誤答と見なす．差分は更新，移動，挿入，削除のすべてを対象とする．延べ利用数は1つの課題の最終提出ソースコードで使用したクラスに関する構文の数に，その課題での提出回数を掛け合わせた値で

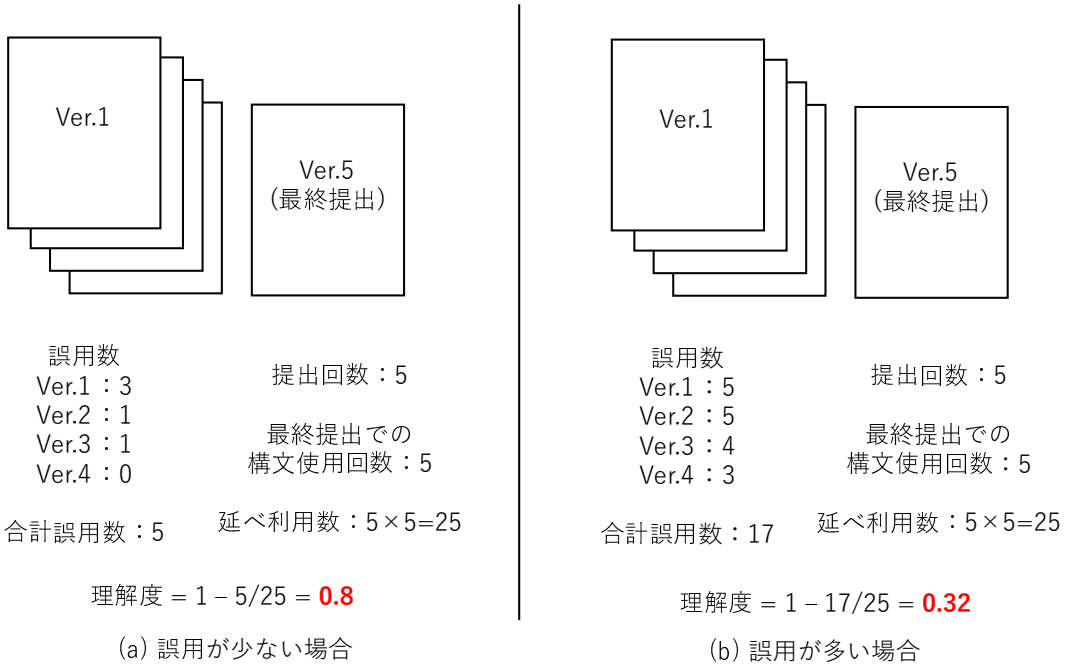

図 3 理解度算出の例

ある．評価対象とする構文集合を最終提出時点のものに固定し，それらの構文が過去の提出において何回正しく使用されていたか評価する．そのため，途中の提出での構文の増減は考慮せず，最終的に使用された構文に対する反復的な使用状況のみを分析対象とする．誤用数を延べ利用数で正規化することで，構文数や提出回数の異なる課題間でも，誤用の発生割合に基づいた理解度の比較をすることができる．

図 3 に理解度の例を示す．(a) は誤用が少ない場合，(b) は誤用が多い場合を表し，いずれの例も提出回数と，対象構文の述べ利用数は同じである．(a) は完答のソースコードと比べて各バージョンでの誤用数が合計 5 回で理解度は 0.8 と高い．誤用数が述べ利用数に対して少ない場合，その構文に対する理解が高いと言える．一方，(b) は各バージョンでの誤用数が合計 17 回で理解度は 0.32 と低い．誤用数が述べ利用数に対して多い場合，その構文に対する理解が低いと言える．本メトリクスは差分数を延べ利用数で正規化することで提出回数やソースコードの規模に影響されず，構文要素単位での理解度を定量的に表現する．

4.3 メトリクスの算出手順

図 4 に理解度メトリクスの算出手順を示す．まず，1) 先行研究の差分フロー抽出システムを用いて，各受講者のある課題における差分フローと，各提出の AST を抽出する．次に，2) 抽出した差分フローの中から，クラスに関する構文を正規表現を用いて抽出し，その差分数をすべて数える．使用する正規表現は以下のよ

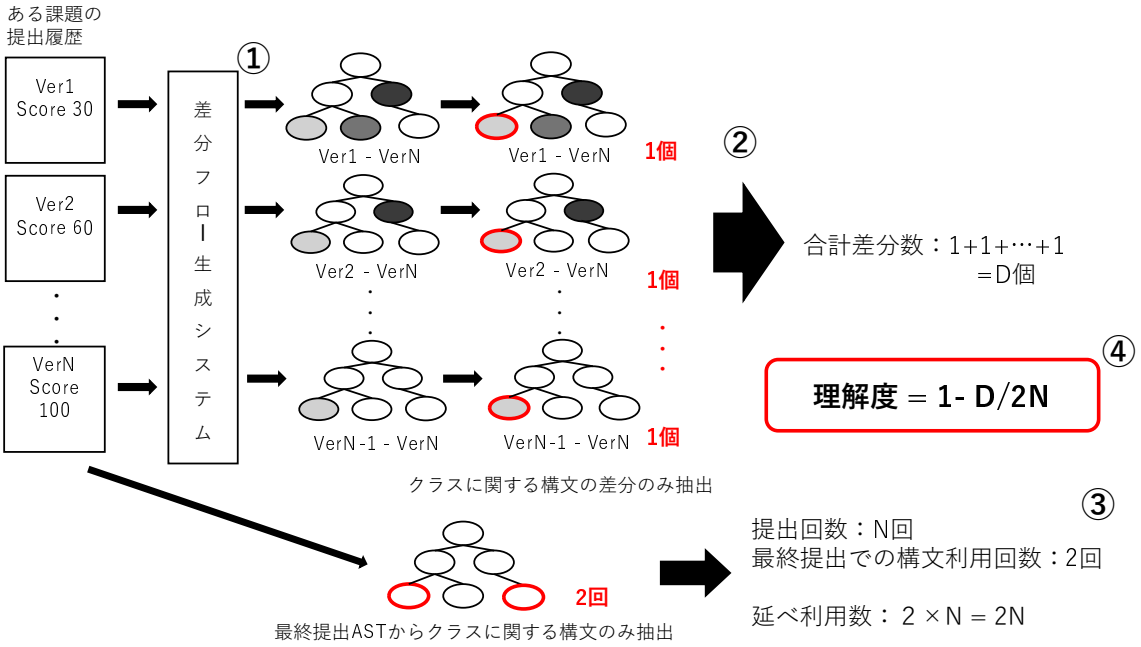

図 4 理解度メトリクスの算出手順

うに設定する．

- インスタンス生成

  `*[\\/](ClassInstanceCreation).*`

- this

  `*[\\/](ThisExpression).*`

- アクセス修飾子

  `*[\\/](Modifier).*`

- メソッド呼び出し部

  `*[\\/]MethodInvocation[\\/]*.*`

- メソッド戻り値の型設定部

  `.*[\\/]MethodDeclaration[\\/].*(SimpleType|PrimitiveType).*`

- メソッドの引数

  `.*[\\/]MethodDeclaration[\\/].*SingleVariableDeclaration.*`

3) 最終提出のソースコードの AST から，クラスに関する構文を 2) と同様に抽出し，その数と提出回数を掛けて延べ利用数を算出する．最後に，4) 算出した合計差分数と延べ利用数から，理解度を算出する．

5 実験

5.1 概要

本研究の提案するメトリクスの有用性を示すことを目的とした実験を行う. 有用性の評価は, 提案手法が算出する理解度と, 受講者による主観評価の理解度との相関を見ることで行う. 提案手法の理解度を算出するためにプログラミング講義中に収集されたソースコードの提出履歴を用いる. 主観評価による理解度を導出するために, 講義を受講する受講者に対してアンケートを行う. この2つの指標に相関があれば, 提案メトリクスは構文要素単位での理解度の有用な指標といえる.

5.2 データセット

変更履歴を抽出するソースコードの提出履歴として, 本校情報工学科の3年生が受講する講義「プログラミングⅡ」で利用しているOJSのソースコード提出履歴を利用する. 本研究はオブジェクト指向プログラミングの重要な概念であるクラスを分析対象とする.「プログラミングⅡ」はJava言語を用いており, 授業で特にオブジェクト指向やクラスの扱い方を中心に学習する. また, 各回の講義内容に沿った課題が出されるため, 提出履歴にはクラスに関する構文が必ず用いられている. ソースコードの提出履歴の取得時期は, 2025年4月16日（水）から2025年10月29日（水）の全19回の講義である. 分析を行うに当たり, 事前に受講者に研究の趣旨を説明し, 同意を得た受講者18名の提出履歴を分析対象とする. 分析対象期間中に出題された計42個の課題に対して, 18名の受講者から981回の提出があった.

クラスに対する主観的な理解度として, 提出履歴を収集した講義を受講した受講者に対するアンケートを用いる. 主観的な理解度評価はそのときの受講者の気分や自己評価の揺らぎによってばらつく可能性があるため, 1回のアンケートでは安定した評価を得ることが難しい. そのためアンケートは3回実施し, その平均値を分析に用いる. アンケートへの回答はいずれも任意とする. アンケートは以下の4項目を4段階の選択式として設定する.

- STA: スタティックメソッドについてどれくらい理解していますか
- INS: インスタンスメソッドについてどれくらい理解していますか
- CON: コンストラクタについてどれくらい理解していますか
- CLA: クラスについてどれくらい理解していますか

スタティックメソッド(STA)はクラスに直接属した, インスタンスを生成せずに実行できるメソッドの一種であり, インスタンスメソッド(INS)との区別が初学者

表2アンケート実施日別　分析対象者回答人数

アンケート実施日	回答者
1回目(6/11)	16
2回目(8/8)	14
3回目(10/22)	18

にとって難しい構文である. コンストラクタ(CON)はオブジェクト生成時の初期化処理を担う要素であり, クラスの基本概念の理解状況が反映されると考えれる. STAとINS,CONは構文要素グループMTCONおよびALLと対応する. クラス(CLA)はクラスに関する構文全体の理解度を把握するために設定する. CLAは構文要素グループCLASSとALLと対応する.

いずれの項目も回答は4段階の選択肢を設定する.

1. 全く理解できていない（何を聞かれているか分からない。または、資料等を見ても、先生や友人に聞いても書けない）
2. あまり理解できていない（資料等を見ても分からず、先生や友人に聞けば書ける）
3. 少し理解している（資料等を見れば書ける）
4. 理解している（資料等を見なくても書ける）

表2に3回行ったアンケート別の分析対象者の回答人数を示す. これより, 1, 2回目のアンケートでは分析対象者18名全員が答えていないことが分かる. そのため, 回答していない受講者の理解度は算出せず, 分析に用いないものとする.

5.3 実験手順

提案手法に基づいて算出したメトリクスとアンケート結果の相関を分析する. 提案手法は利用数／間違い数を数える構文要素を変化させることで, 異なる対象や範囲を分析できる. 本稿では4.1節で説明した3つの構文要素グループを対象としてメトリクスをそれぞれ算出し, 対応する主観評価との相関を分析する. 加えて, 主観評価についても単体項目だけでなく, 複数項目を組み合わせた評価を行う. プログラミングに対する理解は個々の構文や概念が独立して成立するものではなく, 複数の概念が相互に関係する. 例えば, ある構文要素を表面的に理解していても, それと関連するほかの概念を十分に理解していなければ, ソースコード内で適切に利用できない場合がある. そのため, 単一のアンケート項目のみを用いて理解度を評価した場合, 受講者の理解状況を部分的にしかとらえられない可能性がある. 本研究では複数のアンケート項目を組み合わせた相関分析も行い, 関連する概念を統合的に理解できているか評価する.

表3構文要素グループと主観評価の対応表

		構文要素グループ							
		MTCON					CLASS	ALL	
		M1	M2	M3	M4	M5	C1	A1	A2
主観評価	STA	o			o	o			o
	INS		o		o	o			o
	CON			o		o			o
	CLA						o	o	o

表3に本稿で分析を行う構文要素グループと主観評価の対応を示す. M1〜M5はメソッド・コンストラクタに関する分析で, MTCONとの相関を見る. 主観評価のM1〜M3はそれぞれSTA,INS,CONの結果をそのまま用いる. M4とM5は複数の主観評価を組み合わせた分析を行う. M4はSTAとINSを組み合わせる. スタティックメソッドとインスタンスメソッドは, どちらもメソッドの定義および呼び出しに関する概念であり, これらはインスタンスに依存するか否かという違いがある. そのため, この2つを組み合わせるとメソッドという概念を包括的に捉えた理解度として扱うことが可能になる. M5はSTA,INS,CONを組み合わせる. この組み合わせは, クラス内で定義される処理全体に対する理解を評価する目的に適している. そのため, メソッドおよびコンストラクタに関する理解を総合的に捉える指標といえる. 一方コンストラクタは, オブジェクト生成時にのみ呼び出される特殊なメソッドであり, CONとインスタンスメソッドとの区別に対する理解を把握するためのSTAとでは, 意図が異なる. 同様に生成されたオブジェクトに対する振る舞いを定義するインスタンスへの理解状況を把握するためのINSとも意図が異なる. よってCONとINSの組み合わせや, CONとSTAの組み合わせは分析から除外する. 同様に, C1とA1は対応する構文要素と主観評価単体の相関を, A2ではすべての構文要素からメトリクス値を求め, すべての主観評価を組み合わせた値との相関を求める.

複数の主観評価項目の組み合わせによる分析が妥当か判別するために, Cronbach's αを用いる. Cronbach's αは複数の項目から構成される尺度において, それらの項目が同一の概念を測定しているか示す指標であり, 内的一貫性の指標として多く用いられる. Cronbach's αの値は0から1の範囲を取り, 一般に0.7以上で許容可能, 0.8以上で高い内的一貫性を有すると解釈されることが多い[9]. 本研究では質問項目間の一貫性をより厳密に確保するため, しきい値を0.8とし, これを超えた複数項目の組み合わせを分析対象とする.

以上の実験手順を3回分のアンケートに対して実施する. アンケート実施日により近い講義で出された課題が, そのときの受講者の理解度を適切に反映していると考えられるため, 理解度メトリクスの算出にアンケート実施日の直近3回の講義の提出履歴を利用する. 表4にアンケート実施日と, 対応する講義グルー

表 4 アンケート実施日と講義回

	アンケート実施日	講義回	講義実施日
1 回目 (Q1)	6/11	4,5,6	5/7,14,21
2 回目 (Q2)	8/8	11,12,13	7/2,9,16
3 回目 (Q3)	10/22	18,19,20	10/8,15,22

プとその実施日を示す. このとき, Q1 と Q2 ではアンケート実施日と講義実施日の間で期間が空いている. これは, 当該期間中に定期試験が実施され, 講義および課題提示が行われていなかったためである.

表 5 MTCON の理解度と主観評価 M1〜M5 の相関

		Q1	Q2	Q3
M1:STA	r	-0.063	0.487	0.373
	p 値	0.818	0.074	0.125
M2:INS	r	0.019	0.035	**0.467**
	p 値	0.944	0.904	0.049*
M3:CON	r	-0.082	0.043	0.436
	p 値	0.763	0.884	0.068
M4:STA,INS	r	-0.017	0.282	0.436
	p 値	0.949	0.325	0.068
M5:STA,INS,CON	r	-0.050	0.221	**0.470**
	p 値	0.855	0.445	0.047*

6 結果・考察

6.1 主観評価項目間の内的一貫性の検証

8 つの分析のうち，M4, M5, A2 の 3 つでは複数の主観評価を組み合わせて分析を行う．この組み合わせの妥当性を確認するため，Cronbach's α を計算した．M4(STA,INS) では $\alpha = 0.852$, M5(STA,INS,CON) では $\alpha = 0.830$, A2(STA,INS,CON,CLA) では $\alpha = 0.829$ となり，3 つの分析すべてで複数項目の組み合わせを用いることの妥当性を確認できた．

6.2 理解度と主観評価

表 5 に構文要素グループ MTCON の理解度と，主観評価 M1〜M5 の相関係数 r と p 値を示す．太字は有意 ($p < 0.05$) な相関が見られた項目を示す．Q1 はいずれの理解度に対しても相関係数が 0 に近く，相関は見られない．Q2 は M1(STA), M4(STA,INS), M5(STA,INS,CON) で 0.221〜0.487 の正の相関が見られたが，いずれも有意な相関ではなかった．M2(INS) と M3(CON) では相関係数が 0 に近かった．Q3 はすべての理解度において 0.375〜0.470 の正の相関が見られ，M2(INS) と M5(STA,INS,CON) で有意な相関だった．

この有意な相関があった M2(INS), M5(STA,INS,CON) について考察する．まず M5 については，STA, INS, CON といった複数の項目を合算してその平均値を主観評価として用いている．この 3 つの項目は構造や定義・利用の観点で似た概念を持っている．しかし，アンケート回答者の主観や質問への解釈の違いによって，3 項目に対しての回答値が変動することがある．これにより単一の質問では概念全体を測定できない．類似項目を合算し平均を取ることで，偶然的なばらつきや解釈の違いが平均化される．M5 では，STA, INS, CON という同じ概念のアンケート項

表 6 CLASS の理解度と主観評価 CLA の相関

	Q1	Q2	Q3
r	-0.052	-0.265	0.436
p 値	0.848	0.357	0.069

目を平均化したことで偶然誤差が低減され，MTCON の理解度に対して，単一項目よりも安定して相関を取ることが出来たと考えられる．

これに対して M2 では，主観評価 INS が単一項目であるにもかかわらず，有意な相関が認められた．これは，構文要素グループ MTCON で抽出している構文要素が関係していると考えられる．MTCON は，メソッド呼び出し部，戻り値の型設定部，引数部の 3 つの構文要素が含まれるが，これらの構文要素は主にインスタンスメソッドを利用するときに多く見られる．そのため，抽出した差分もインスタンスメソッドを利用したときの差分の数が多くなる．INS ではインスタンスメソッドをどのくらい理解しているか尋ねているため，インスタンスメソッドの差分数の比率が多くなる MTCON の理解度は，この INS に対して，STA, CON よりも高い相関が出てかつ有意性もみられたのではないかと考えた．また，この 3 つの構文要素のうち，呼び出し部と引数部についてはコンストラクタを利用するときにも見られるため，インスタンスメソッドの次に差分数も多くなり，M3(CON) では相関が見られたと考えられる．そして，スタティックメソッドについてはインスタンスメソッドとコンストラクタに比べて，この構文要素から取れる差分数が少なかったため，弱い相関となったと考えられる．

表 6 に構文要素グループ CLASS に属する C1 を対象に算出した理解度と，主観評価 CLA の相関係数 r と p 値を示す．Q1 は相関係数が 0 に近く，相関は見られなかった．Q2 で弱い負の相関，Q3 で正の相関が見られたが，いずれも有意な相関は見られなかった．CLASS はクラスの定義・利用に関する構文要素グループであり，抽出する構文要素は，インスタンス生成と this, アクセス修飾子である．これに対して，主観評価 CLA では「クラスについてどれくらい理解しているか」を尋ねており，このクラスへの理解は，クラスを正しく定義・利用できているか測る CLASS の理解度と必ずしも一致するわけではない．また,「クラスについてどれくらい理解しているか」という質問が抽象的で，回答者によって解釈が異なる場合がある．よって，CLASS の理解度と主観評価 CLA の間では有意な相関がみられなかったと考える．

表 7 に構文要素グループ ALL に属する A1 と A2 を対象に算出した理解度と，主観評価の相関係数 r と p 値を示す．他の理解度と同様に，Q1 と Q2 ではほとんど相関は見られず，Q3 で A1(0.436) と A2(0.393) それぞれに正の相関が見られた．A2(STA, INS, CON, CLA) については p 値が 0.044 と有意な相関だった．

有意な相関がみられた A2 については，M5 と同様に複数の項目を合算してその

表 7 ALL の理解度と主観評価 A1〜A2 の相関

		Q1	Q2	Q3
A1:CLA	r	-0.008	-0.276	0.393
	p 値	0.976	0.337	0.104
A2:STA,INS,CON,CLA	r	-0.05	0.031	**0.477**
	p 値	0.841	0.916	0.044*

平均値を主観評価として用いている．STA,INS,CON,CLA の全ての項目が，クラスという概念に関係がある．よって合算して平均化をすることで，クラスに関する構文全体の理解度を算出した ALL に対して，CLA という単一項目を用いた A1 とは異なり．有意な相関が得られたと考える．

6.3 Q1，Q2 の相関についての考察

分析結果から，MTCON, CLASS, ALL の全ての構文要素グループで，算出した理解度と Q1, Q2 の主観評価とではほとんど相関がみられず, 相関があったとしても有意性が見られることはなかった．しかし Q3 の主観評価との相関を見ると，どの構文要素グループにおいても，相関がみられた．これは，課題の難易度が関係していると考えられる．Q1, Q2 を実施した時期の講義第 4〜6 回，11〜13 回で出された課題は，クラスの基本概念やインスタンスメソッドなどのオブジェクト指向の初歩的な内容が中心だった．これらは，受講者自身のクラスに対しての理解･未理解に関わらず, 講義資料に書かれていることを書けば正答するような難易度であったため，理解している受講者としていない受講者の差分数に大きな差が生まれなった．しかし，Q1 を実施した時期の講義 18〜20 回で出された課題は，継承や配列クラスなどの応用やファイル操作も含まれてくることで，講義資料に書かれている通りのコードを書いても正答にならないような難易度の問題が見られる．これによって，クラスについてあまり理解していない受講者は，段々誤答を繰り返し，差分数も多くなる．そして，理解している受講者との差が大きくなり，アンケートとの相関がみられるようになったのではないかと考える．

7 おわりに

本研究ではOJSに記録された受講者の提出履歴からクラスに関する構文についての差分フローを抽出し，差分数や差分内容から受講者のクラスに関する構文理解度を示すメトリクスを提案した．メトリクス算出には，インスタンス生成，this, アクセス修飾子，メソッド呼び出し，メソッド戻り値の型設定，メソッドの引数の6つの構文要素が含まれる差分を抽出しその差分数を用いた．実験では算出した理解度メトリクスの有用性を示すために，受講者によるクラスに関する構文理解度の主観評価との相関を分析した．実験の結果，クラスについて理解していない受講者が誤答を繰り返す3回目のアンケートにおいて，複数の構文要素グループで有意な相関が確認された．提案手法はクラスに関する構文について構文要素単位で受講者の理解度を定量的に示すことができ，この理解度が有用であるといえることがわかった．

本研究の今後の発展として，提案の理解度をif文やfor文，その他構文に対しても適用することで構文要素単位での理解度を定量化する範囲を拡大することが挙げられる．理解度は1つの構文要素から算出するのではなく，関係がある複数の構文要素を組み合わせて算出することでより適切な理解度になると考えられる．また，if文やfor文の様な複数の要素から構成される構文要素に対しては，構成要素ごとの理解度を合算することで，if文全体やfor文全体の理解度が算出できる．本研究が提案する理解度は，受講者の提出履歴から自動的に算出可能であることから，受講者への学習フィードバックや教員の指導支援への活用が期待される．受講者に対して理解度を提示することで，自身の理解状況を客観的に確認できる．これにより，自分の理解できていない部分について調べたり，過去の課題を振り返って理解していない部分を復習したりして，理解を高めることが出来る可能性がある．教員にとっては，クラスに関する構文への理解が不十分な受講者を把握する指標として利用でき，より受講者に適した指導が可能になると考えられる．例えば，メソッド･コンストラクタに関する構文の理解度指標が低い受講者に対して，メソッドの定義部分や呼び出し部分に焦点を当てた補助教材を提示するなど，構文要素に応じた個別の指導を容易に行うことが可能になる．

また，本研究で算出した理解度は，完答に至るまでの誤用数によって算出した．これは厳密には完答に至るまでどのくらい苦労したかを示しており，完答時点での理解度とは少し異なる．完答時点での理解度を算出する手法として，類似課題を2回行い，各回にて本研究で提案した理解度を算出し，その理解度の差によって理解度合いを求める方法が挙げられる．そして，1回目の課題の後に行ったアンケートと相関を見ることで，改めて完答時点での理解度の有用性が評価できると考えられる．

謝辞

本研究を行うにあたり, 常日頃から熱心にご指導いただき, 多くの貴重なご助言を賜りました指導教員の上野秀剛准教授に深く感謝申し上げます. また, 本研究に関して有益なご意見をいただきました内田眞司教授, 岡村真吾教授, 岩田大志准教授に厚く御礼申し上げます.

参考文献

[1] 青木晃汰, 上野秀剛, ”差分構文木を用いたプログラミング授業受講者のコーディング特徴の自動抽出”, FIT2023 講演論文集, pp.21-28 (2023).

[2] 薮中天空, ”プログラミングスキル表現手法としてのスキルツリーの提案”, 奈良工業高等専門学校情報工学科令和6年度卒業研究論文 (2025).

[3] 三野天羽, 上野秀剛, ”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”, 信学技報教育工学研究会, Vol.121, No.232, pp.31-36 (2021).

[4] 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 (2022).

[5] Yana Malysheva, Caitlin Kelleher, ”An Algorithm for Generating Explainable Corrections to Student Code”, Koli Calling'22: Proceedings of the 22nd Koli Calling International Conference on Computing Education Research, No.13, pp.1-11 (2022).

[6] 鄭佳健, 寺田実, ”初心者のプログラムを正誤と質で評価するシステム”, 第62回プログラミング・シンポジウム予稿集, Vol.2021, pp.75-82 (2021).

[7] 涌井慧, 寺田実, 中山泰一, ”Path Context を用いたプログラム部分点の算出方式とその評価”, 情報教育シンポジウム論文集, Vol.2024, pp.118-123, (2024).

[8] 漆原宏丞, 本多佑希, 岸本有生, 兼宗進, ”構文木を利用したプログラミング課題の理解度判定方式”, 第84回全国大会講演論文集, Vol.2022, No.1, pp.633-634, (2022).

[9] Joseph A. Gliem, Rosemary R. Gliem, ”Calculating, Interpreting, and Reporting Cronbach's Alpha Reliability Coefficient for Likert-Type Scales”, Proceedings of the 2003 Midwest Research-to-Practice Conference in Adult, Continuing, and Community Education, pp.82-88 (2003).