博士論文

Javaプログラミング学習支援システムのソフトウェアアーキテクチャと2種類の新問題形式の提案

平成30年2月

石　原　　信　也

岡山大学大学院
自然科学研究科

Javaプログラミング学習支援システムのソフトウェアアーキテクチャと2種類の新問題形式の提案

要旨

オブジェクト指向プログラミング言語 *Java* は，開発環境，堅牢性，可搬性に優れており，エンタープライズシステムから組み込みシステムに至るまで，産業界で広く利用されている．そのため，Java 言語技術者の育成が強く求められており，実際，大学や専門学校などの多くの教育機関で，Java プログラミング教育が行われている．

岡山大学分散システム構成学研究室（以下，本グループ）では，Java プログラミング教育の支援を目的として，Web を用いた *Java* **プログラミング学習支援システム** *JPLAS(Java Programming Learning Assistant System)* を提案している．JPLAS では，教員の負担を軽減しながら，学生のプログラミング学習を容易とするため，解答の自動採点を可能とする 2 種類の問題形式，**エレメント空欄補充問題**，**コード作成問題**を提供している．前者では，エレメント単位で空欄としたソースコードを与え，学生に正しいエレメントの補充を求める．解の正誤判定は，空欄前のエレメントとの**文字マッチング**により行う．後者では，動作の正しさを検証するための**テストコード**を与え，学生にそれをパスするソースコードの作成を求める．解の正誤判定は，テストコードを用いた**ソフトウェアテスト**で行う．

現状の JPLAS には，いくつかの問題点が指摘されている．まず，JPLAS の実装において，問題形式毎に，異なる学生がそれまでのシステムをコピーすることでコード作成を進めたために，URL やデータベースを個別に有する，独立した Web システムとなっている．その結果，JPLAS のコードやデータは冗長性が高く，見通しの悪い実装となっており，新しい問題形式の実装が非常に困難となっている．また，現状の 2 種類の問題形式では，難易度の差が大きく，前者の問題が解けても，後者の問題に手が付けられないと言った状況が発生している．

以上の問題点の解決のために，本研究では，まず，様々な問題形式に対応可能な実装とするために，JPLAS の**ソフトウェアアーキテクチャ**を提案する．従来の JPLAS 実装で明らかとなっている，各問題で必要な機能（問題生成，問題提示，採点，結果表示）の整理を行った上で，それらを複数の問題形式に対応可能となる実装方法を採用した．具体的には，各問題で必要な情報の種類を示す**タグ**を定義し，それを用いたテキストファイルで各問題のデータを表現した．また，*Ajax* を用いて動的に画面更新を行うことで，必要なコード量の削減を図った．これにより，従来の JPLAS 実装と比較し，コード量を半分以下に削減した．

次に，既存の 2 種類の問題形式の難易度差を埋めるための新たな問題形式として，**ステートメント空欄補充問題**と**コードクローン除去問題**を提案する．前者の問題では，ステートメント単位で空欄としたソースコードを与え，学生に正しいステートメントの補充

を求める．エレメント単位と異なり，一般に解が一意には決まらないため，テストコードを用いた正誤判定を行う．問題生成時の適切な空欄ステートメントの選択のために，オブジェクト指向言語に拡張した，ソースコードの *PDG* **グラフ**を求め，その次数を用いた．後者の問題では，**コードクローン**（冗長なコード部分）を有するソースコードを与え，4種類の手法のいずれかを用いてコードクローンを除去したコードの作成を求める．解の正誤判定は，コードクローン有無の検査とソフトウェアテストで行う．コードクローン除去手法の理解の支援のために，3段階学習法を提案した．

最後に，本研究の今後の課題について述べる．まず，JPLASの実装において，画像ファイルを含む問題文やJavaバイトコードによる解答提出といった新しい機能への対応が挙げられる．また，コードクローン除去問題において，現在，1種類のコードクローン除去手法にのみ，3段階学習法の実装・評価を終えており，残る2種類の手法への実装・評価が挙げられる．

関連発表論文

1. 学術論文誌

1-1 **Nobuya Ishihara**, Nobuo Funabiki, Minoru Kuribayashi, and Wen-Chung Kao, "A Software Architecture for Java Programming Learning Assistant System," International Journal of Computer & Software Engineering, Vol. 2, No. 1, September 2017.

1-2 **Nobuya Ishihara** and Nobuo Funabiki, "A proposal of statement fill-in-blank problem in Java programming learning assistant system," Information Engineering Express, Vol. 1, No. 3, pp. 19-28, September 2015.

2. 国際会議　会議録

2-1 **Nobuya Ishihara**, Nobuo Funabiki, Tana, and Wen-Chung Kao, "An extension of statement fill-in-blank problem in Java programming learning assistant system," The 4th IEEE Global Conference on Consumer Electronics (GCCE2015), pp. 354-358, October 27 - 30, 2015. (Osaka International Convention Center)

3. 学術研究集会

3-1 **石原信也**, 舩曳信生, 栗林稔, "Java プログラミング学習支援システムのコードクローン除去問題におけるメソッド生成課題の改善," 信学技報, Vol. 117, No. 248, SS2017-25, DC2017-24, pp. 25-30, Oct. 2017.

3-2 **石原信也**, 舩曳信生, 栗林稔, "Java プログラミング学習支援システムにおけるコードクローン除去問題の提案," 信学技報, Vol. 116, No. 426, MSS2016-65, SS2016-44, pp. 47-52, Jan. 2017.

3-3 **石原信也**, 舩曳信生, 栗林稔, "Java プログラミング学習支援システム JPLAS の MVC モデルに沿ったコード記述ルールの提案とその再構築," 信学技報, Vol. 116, No. 85, ET2016-16, pp. 47-52, June 2016.

3-4 **石原信也**, 舩曳信生, "Java プログラミング学習支援システムのコードの記述ルールの検討," 信学技報, Vol. 114, No. 82, ET2014-18, pp. 57-62, June 2014.

3-5 **石原信也**, 舩曳信生, 中西透, "Java プログラミング学習支援システムにおけるステートメント補充問題機能の実装," 信学技報, Vol. 113, No. 482, ET2013-98, pp. 35-40, March 2014.

目 次

図 目 次

表 目 次

コード目次

第1章　はじめに

1.1　Javaプログラミングの特徴

オブジェクト指向のプログラミング言語 Java は，開発環境，堅牢性，可搬性などに優れ，エンタープライズシステムから組み込みシステムに至るまで，産業界で幅広く継続的に利用されている．そのため，産業界から Java 言語技術者の育成が強く求められており，実際，大学や専門学校など多くの教育機関で Java プログラミング教育が行われている．

Java 言語は，C 言語に似た文法を有することから，一般に，C を学習した人には学習しやすい言語であるとされる．それにも関わらず，Java の学習は，C の学習とは違った難しさがある．その一つは，**オブジェクト指向**である．Java では，複数のクラスを用いてコードを構成することからきている．C の学習を終えた人が Java を学習しようとする場合，このような C から拡張された要素が学習上のネックとなり得る．

この C 言語からの拡張要素には，演算子においていくつか存在する．例えば，変数の大きさを調べる *sizeof* **演算子**は C でも定義されているが，Java ではそれに加え，**オブジェクト（インスタンス）**が，指定したクラスまたはその上位のクラスに属しているかどうかを調べる *instanceof* **演算子**が定義されている．

また，C では，その実現したい仕様（アルゴリズム）の実装を，基本的には，構造化プログラミングに沿ったフローチャートと領域図で机上のトレースが可能である．Java では，それに加えて，**継承**を通じた**多態性**や**多義性**を駆使し，**クラス間連携**で実装されることが一般的である．

一方，C や Java の従来のプログラミング教育では，学生に対して，有効とされている**コードリーディング**に対する学習が明示的に行われていないといった問題点が指摘されている [1]．コードリーディングは，新規のコード作成に必要なコードスタイル（書き方）を，既存のコードを読み下すことで理解する学習であるといえる．長期間・広範囲の使用により蓄積された良質なコードの特徴を学ぶことで，既存のコードに含まれる仕様外のコードを検出しセキュリティを向上させる面でも重要性を増している．

Java プログラミング教育では，以上の Java プログラミングの特徴を考慮した，実践的な教育を進めることが重要である．

1.2　Javaプログラミング学習支援システム JPLAS

本グループでは，この Java プログラミング教育での学習支援を目的として，Web を用いた *Java* **プログラミング学習支援システム** *JPLAS(Java Programming Learning Assistant*

System) を提案している [2]-[4]．JPLAS では，教員の負担を軽減しながら，学生の自宅などでのプログラミング学習を容易とするために，Web サーバにおいて，学生からの解答のオンライン自動採点を行うことを基本としている．学生は，Web ブラウザを用いて JPLAS サーバにアクセスし，問題の閲覧や解答を行うことができる．

JPLAS では，Java プログラミング教育の進捗や学生の理解度に応じた学習環境を提供するために，2 種類の問題を用意している．1 つは， 模範となる Java ソースコードの中から学習の対象となる語を空欄語として与え，そこに正しい語のスペルを解答する**エレメント空欄補充問題**である [3][4]．本問題では，Java の文法などを学ぶ初学者を対象としている．もう 1 つは，**テスト駆動型開発手法**により，学生の解答コードの自動検証を行う**コード作成問題**である [2]．ここでは，学生は，JPLAS で提示される**テストコード**（テスト用 Java コード）の実行でエラーを検出しない，Java ソースコードの作成が求められる．本問題では，文法学習を終え，クラス全体を含むコードの作成を学ぶ学生を対象としている．

現状の JPLAS には，いくつかの問題点が指摘されている．まず，JPLAS の実装において，問題形式毎に，異なる学生がそれまでのシステムをコピーすることでコード作成を進めたために，URL やデータベースを個別に有する，独立した Web システムとなっている．その結果，JPLAS のコードやデータに冗長性が高く，見通しの悪い実装となっており，新しい問題形式の実装が非常に困難となっている．また，現状の 2 種類の問題形式では，難易度の差が大きく，前者の問題が解けても，後者の問題に手が付けられないと言った状況が発生している．

1.3　本研究の目的

本研究では，上記の問題を解決するために，JPLAS の**ソフトウェアアーキテクチャ**の提案と，既存の 2 種類の問題形式の難易度差を埋めるための新たな問題形式として，**ステートメント空欄補充問題**と**コードクローン除去問題**を提案する．

1.3.1　JPLAS ソフトウェアアーキテクチャ

まず，様々な問題形式に対応可能な JPLAS の**ソフトウェアアーキテクチャ**の提案を行う．従来の JPLAS の実装で明らかとなっている，各問題で必要な機能（問題生成，問題提示，採点，結果表示）の整理を行った上で，それらを複数の問題形式に対応可能となる実装方法を採用する．具体的には，各問題で必要な情報の種類を示す**タグ**を定義し，それを用いたテキストファイルで各問題のデータを表現する．また，XMLHttpRequest により，Web ブラウザとサーバ間でページ遷移を伴わない HTTP 通信を可能とする，*Ajax* **技術** [12] を採用する．これにより，動的に画面更新を行うことで，必要なコード量の削減を可能とする．これにより，従来の JPLAS 実装と比較し，コード量を半分以下に削減できたことを示す．

1.3.2　ステートメント空欄補充問題の提案

次に，ステートメント単位で空欄としたソースコードを与え，学生に正しいステートメントの補充を求める**ステートメント空欄補充問題**を提案する．本問題では，エレメント単位で空欄とする**エレメント空欄補充問題**と異なり，一般に解が一意には決まらないため，**テストコード**を用いた解答の正誤判定を行う．

本問題生成時の適切な空欄ステートメントの選択のために，対象となるコードを分類し，メソッドのステートメントが空欄対象となる場合には，ソースコードのプログラム依存グラフ (以下，*PDG* **グラフ**と記す) を求め，その次数の高い順に，空欄ステートメントを選択することとした．PDG グラフでは，ステートメントを点で表し，依存関係を有する 2 点（ステートメント）間に辺を設ける．この時，Java といったオブジェクト指向言語では，変数に加え，オブジェクト間の依存関係も重要となるため，PDG グラフの辺には，それも含める．このオブジェクト間の依存関係は，ドット演算子を調べることで検出する．

ステートメント補充問題による学習効果を確認するため，学生を対象に, 実際に本機能を使ってもらった. その結果, 類似した問題について解答速度が向上していた．次に，5 つの Java コードに適用して抽出されたコアステートメントを，Java の学習者による主観的な抽出結果と比較した．その結果，両者に高い一致がみられ，提案アルゴリズムの有効性が確認された．

1.3.3　コードクローン除去問題の提案

更に，ソースコード中の全く同じ，あるいは，類似したコードの断片**コードクローン**を有するソースコードを与え，そこからコードクローンを除去したコードの作成を求める**コードクローン除去問題**を提案する．リファクタリング技法としてまとめられているコードクローン除去の手法を，プログラミング学習の観点から 4 種類のコードクローン除去問題に分類する．解の正誤判定は，コードクローン有無の検査とソフトウェアテストで行う．また，コードクローン除去手法の理解の支援のために，3 段階学習法を提案する．

生成した 4 種類のコードクローン除去問題の例題を学生に解答してもらい, 難易度の評価をおこなったところ，理解支援が必要なギャップがあった．その理解支援のために 3 段階学習法を提案し，同様の評価を行った．

1.4　本論文の構成

本論文では，以下の構成に従って，Java プログラミング学習支援システムのソフトウェアアーキテクチャと 2 種類の新しい問題形式に関する研究成果の報告を行う．

第 2 章では，従来の Java プログラミング学習支援システム JPLAS の機能の概要とその実装方法を紹介する．

第 3 章では，JPLAS の各機能を実現する，JPLAS ソフトウェアアーキテクチャの提案を行う．ここでは，提案するコード記述ルールを示した後に，その実装結果を示す．続いて，本実装による JPLAS の機能拡張性についての評価を行い，最後に，本章のまとめと今後の課題を述べる．

第 4 章では，JPLAS の新しい問題形式として，ステートメント空欄補充問題を提案する．Java 言語の学習テーマを大きく 3 要素に分類し，それぞれに従ったコアステートメント抽出法を提案し，検証する．最後に，本章のまとめを述べる．

第 5 章では，コードクローン除去問題機能の提案を行う．既存のリファクタリング技法を参照しながら 4 種類の問題を定義し，学生が各問題を理解するための 3 段階学習法を提案する．次に，コードクローン除去問題の解の正誤判定のためのテストコードを提示し，評価のための適用結果を示す．最後に，本章のまとめと今後の課題を述べる．

第 6 章では，本研究の関連研究を示す．

最後に，第 7 章では，本研究のまとめを行い，今後の課題について述べる．

第2章　従来のJavaプログラミング学習支援システム

本章では，本グループが提案している，従来の Java プログラミング学習支援システム JPLAS の概要と，利用手順について述べる．

2.1　はじめに

JPLAS では，異なる学習レベルに対応するために，**エレメント空欄補充問題**と**コード作成問題**の 2 種類の問題が提供されている．

エレメント空欄補充問題では，エレメント単位で空欄化された Java のソースコードを提示し，その空欄に該当する語（エレメント）の解答が求められている．本問題は，Java プログラミング初学者の学習支援を目的としている．空欄化されるエレメントは，**予約語**，**識別子**，**文法記号**，**比較演算オペレータ**であり，解の一意性を充たすエレメントを選択するため，グラフ理論を用いた，**空欄エレメント選択アルゴリズム**を提案している [?]．解答の正誤判定は，JPLAS サーバにおいて，予め登録されている正解との文字マッチングで行われる．学生に，判定結果を直ちにフィードバックすることで，解答の誤りの早期発見とその修正が可能となる．

コード作成問題では，課題文とテストコードを提示し，それらの要件を充たす Java のソースコードの作成が求められている．本問題は，より実践的な Java プログラミングの学習支援を目的としている．解答のソースコード（**解答コード**）の正誤判定は，JPLAS サーバにおいて，検証ツール *JUnit*[13] 上で**テストコード**を用いて行われる．学生に，検証結果を直ちにフィードバックすることで，解答コードの誤りの早期発見とその修正が可能となる．学生は，正しい検証結果が得られるまで，解答コードの修正と提出を繰り返すことで，Java プログラミングの学習を進める．本問題は，**テスト駆動開発手法**に基づいて構築されている．

JPLAS では，2.1 に示す，教員による問題作成・登録，学生の学習結果の把握，および，学生による問題の閲覧，解答の作成・提出，正誤判定結果の閲覧といった機能が必要となる．従来の JPLAS では，問題毎に，それらの機能を独立して実装している．その結果，新しい問題形式や機能の追加・修正に関して，見通しの悪い実装環境となっている．

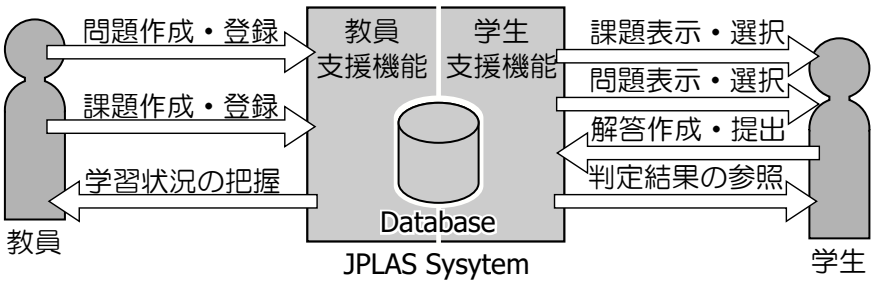

図 2.1: JPLAS の機能

2.2 JPLAS の機能概要

従来の JPLAS は，図 2.1 に示すように， **教員支援機能**と**学生支援機能**で構成される．前者では，問題作成・登録，学習状況把握のサービスが提供されている．後者では，課題選択，問題参照，正誤判定，結果参照のサービスが提供されている．

JPLAS の教員，学生の両支援機能の全体的な利用手順は以下のとおりである．

1. 教員による問題作成と登録
2. 教員による課題作成と登録
3. 学生による課題の選択
4. 学生による問題の選択
5. 学生による問題の解答作成と提出
6. 学生による解答の正誤判定結果の参照
7. 教員による学生の学習状況の把握

2.2.1 教員支援機能

ここでは，教員支援機能で提供されているサービスにおける，教員の手順について述べる．

1. **問題作成・登録**
   教員は，問題の使用に適切な Java のソースコード（サンプルコード）を，データベースに登録する．教員は，これらを用いて，新たな問題を作成し，データベースに登録する．
2. **課題作成・登録**
   教員は，授業毎の学習目標を明確にするために，データベースに登録されている**問題**を，グループ化することで，**課題**を登録する．学生は，課題単位で学習を進める．

3. **学習状況把握**
   教員は，学生の JPLAS での学習進度を把握するために，全学生に対する各問題の解答状況を閲覧する．

2.2.2 学生支援機能

ここでは，学生支援機能で提供されているサービスにおける，学生の手順について述べる．

1. **課題表示・選択**
   学生は，教員から提示されている JPLAS の課題の一覧を閲覧し，その中から解答する課題を選択する．

2. **問題表示・選択**
   学生は，選択した課題に含まれている問題の一覧を閲覧し，その中から解答する問題を選択する．

3. **解答作成・提出**
   学生は，提示された問題に対する解答を用意されたフォームに記入する．記入終了後，提出のボタンをクリックすることで，解答がサーバに送信される

4. **正誤判定結果の参照**
   学生は，今回の解答に対するサーバでの正誤判定結果を閲覧する．

2.2.3 エレメント補充課題の正誤判定

エレメント補充課題の正誤判定は，空欄ごとに，学生の解答の語を，データベースに保存されている正解の語との文字マッチング (逐語比較) で行う．学生への正誤判定結果の提示では，各空欄に対して，その背景色を変えることで正誤を示す（正答は白，誤答はピンク）．その問題の評点として，全空欄中の正解数の割合（百分率）を提示する．学生から解答が提出される度に，データベースに，提出された解答とこの評点が保存される．

2.2.4 コード作成問題の正誤判定

学生からのソースコード（解答コード）の正誤判定は，JUnit [13] 上で，テストコードを用いたソフトウェアテストにより行われる．テストコードでは，*assert* で始まる**テストメソッド**の実行により，解答コードの実行結果（実行値）が，課題での要求結果（期待値）に一致しているか否かによって，課題の様々な仕様の正誤判定がなされる．その問題の評点として，全テストメソッド中の正解メソッド数の割合（百分率）を提示する．学生から解答が提出される度に，データベースに，提出された解答コードとこの評点が保存される．

なお，テスト駆動開発手法の枠組みやテストコードの理解は，必ずしも学生にとって容易ではない．そのため，テストコード自体の学習問題も作成されている．

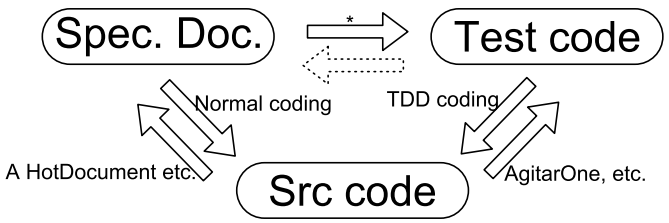

図 2.2: テスト駆動開発

2.3 従来の JPLAS の実装環境

2.3.1 ソフトウェア

図 2.3 に JPLAS の実装環境を示す．OS として採用した Ubuntu は仮想デスクトップ VMware 上で動作し，運用時には大学で指定された IP アドレス/MAC アドレスを通じてクライアントの Web ブラウザと通信する．Web サーバとして，JSP/サーブレットのコンパイラである Tomcat が JVM 上で使用される．仮想環境を採用したのは，様々なサーバ環境への移植を容易とするためである．

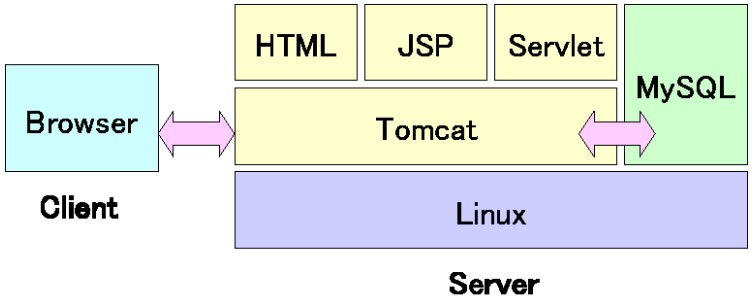

図 2.3: 従来の JPLAS の実装環境

学生はブラウザから JPLAS へアクセスする．ブラウザにより機能に差があるため，現状，JPLAS では *Firefox* ブラウザのみ対応している．

JVM は Java の実行環境であり，OS の違いを吸収する働きがある．

Tomcat はブラウザへデータを送信するサーバであり，以下に述べる JSP とサーブレットのコンパイラでもある．

HTML は，Tomcat によりブラウザに送信されるデータである．*HTML* では，Web 標準においては，データの構造あるいはデータの本体を記述する．その装飾は，カスケーディングスタイルシート *CSS* で記述され，また，その動的な機能は，*JavaScript* が実行している．このように，通常，3 層構造となっている．

JSP (Java Server Pages) ファイルは，HTML 内に Java のコードを埋め込んだものであり，これによって Tomcat は動的に Web ページを生成してクライアントに返すことができる．**サーブレット**は，Java で作成されたプログラムであり，Web ページなどを動的に生成したり，データ処理を行ったりするために使用される．

Tomcat サーバからは特定の場所に配置されたコンパイル済みの *Java* **バイトコード**を呼び出すことができる．

問題データや解答データを管理するためのデータベースには，*MySQL* を採用している．これらのデータの蓄積は，MySQL データベースへの連携として実装される．

2.3.2　従来のシステムモデル

Web アプリケーションを構築するためには，システムの構成を，ビジネスロジックを実装したモデル部分 (**M**odel) と, ユーザインターフェース（**V**iew）部分，更に両者をコントロールする部分（**C**ontrol）に三つに分割する MVC モデルをフレームワークとして使用するのが一般的である．

図 2.4 に，従来の JPLAS 実装に使用された MVC モデルを示す [16]．これは，*Struts2*[17] などでも使われている開発フレームワークである．

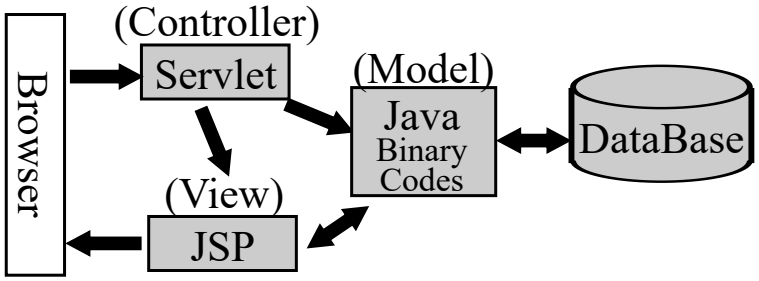

図 2.4: Languages for MVC model in existing JPLAS.

図 2.4 において，ユーザは Web ブラウザからサーブレットを呼び出す．サーブレットは Java のバイトコードと連携して処理を行い，その処理結果を JSP を用いて Web ページに整形し，ブラウザへ発信する．このフレームワークでは，ページ遷移が前提となるため，メニューといった共通部分に冗長な処理が必要となる．

2.4　従来の JPLAS の問題点

JPLAS は，学習レベルの異なる様々な学生に対応するため，難易度の異なる 2 種類の問題形式（**エレメント補充問題**，**コード記述問題**）を提供している．しかし，その難易度レベルの差が大きく，前者の問題が解けても，後者の問題に手がつかないといった問題が生じている．それらの問題間の難易度差を埋める，新たな問題形式の提示が JPLAS に必要である．

また，JPLAS のコード実装は，本研究室に所属した，年度の異なる複数の学生によって，初期の学生が開発したコードを元に，それ以降の年度毎の学生が，それぞれの研究テーマに基づいた，新機能の追加実装を行うことで，継続的に行われてきた．これにより，大学における実践的なプログラミング開発を学ぶための機会ともなっているが， JPLAS

のコードには，ほぼ同一の機能を有するクラスやメソッドが多く存在するといった，冗長で見通しが悪く，新たな機能の拡張が困難となっている．JPLAS に適したソフトウェアアーキテクチャを提示し，それに基づいた実装を行うことで，その改善が必要である．

2.5 おわりに

本章では，２種類の問題を中心として，従来の Java プログラミング学習支援システム JPLAS の機能および実装の概要とその問題点を紹介した．

第3章　ソフトウェアアーキテクチャの提案

本章では，従来の JPLAS の実装における問題点の解決を目的として，JPLAS 全体を*MVC*モデルに沿った実装するためのソフトウェアアーキテクチャの提案とその実装を行う．

そのために， コード記述ルールの提案とその再構築を行う．本ルールでは，コード量の最小化のため，データベース処理のための抽象クラスの採用， レスポンシビリティ・チェーンデザインパターンを用いた異なる正誤判定機能の実装，JSP + Ajax による画面遷移の実現，などを規定した．本提案の評価のために，再構築前後の JPLAS のファイル数の比較， 2 種類の新機能の拡張での追加ファイル数の評価を行う．

3.1　はじめに

本章では, 様々な問題形式に対応可能な JPLAS のソフトウェアアーキテクチャの提案を行う．従来の JPLAS の実装で明らかとなっている，各問題で必要な機能（問題生成，問題提示，正誤判定，結果表示）の整理を行った上で，それらを複数の問題形式に対応可能となる実装方法を採用する．具体的には，各問題で必要な情報の種類を示すタグを定義し，それを用いたテキストファイルで各問題のデータを表現する．

また，XMLHttpRequest により，Web ブラウザとサーバ間でページ遷移を伴わない HTTP 通信を可能とする，*Ajax* 技術 [12] を採用する．これにより，動的に画面更新を行うことで，必要なコード量の削減を可能とする．これにより，従来の JPLAS 実装と比較し，コード量を半分以下に削減できたことを示す．

3.2　問題ファイルのタグ

提案する JPLAS のソフトウェアアーキテクチャでは，JPLAS で提供する，様々な問題形式に対して，1 つのテキストファイル（問題ファイル）で， 1 つの問題で必要となるデータを記述可能とする．そのため，本研究では，問題ファイル中に，問題毎に必要となるデータ項目を示すタグとして，表 3.1 の文字列を提案する．

3.2.1　実装した学生支援機能

今回，本研究では，JPLAS のソフトウェアアーキテクチャに基づき，学生支援機能のみを実装した．を示す．図 3.1 に，本機能の処理の概要を示す．

表 3.1: 問題ファイルのタグ

マーカ	意味
//@JPLAS answer	後置する文は正解単語リスト
//@JPLAS statement	後置する文は仕様を表す課題文
//@JPLAS test code	後置する文はテストコード
//@JPLAS output	後置する文は出力例
_ (下線)	単語補充のための空欄位置
なし	問題コード

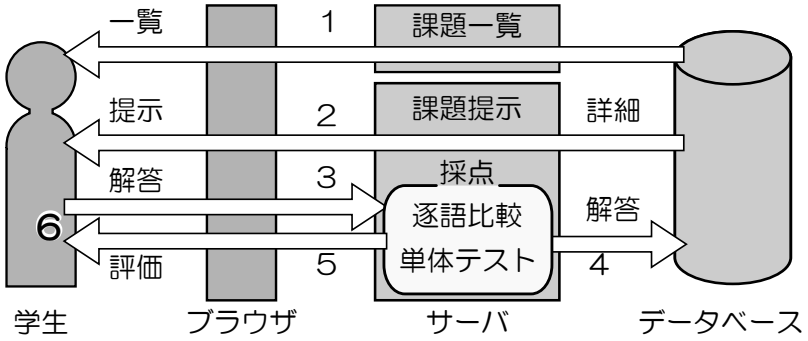

図 3.1: JPLAS の処理の流れ

学生支援機能における処理の流れは，以下となる．

1. 学生からのアクセスに対し，JPLAS は，データベースからその学生に出されている課題の一覧を表示する．
2. 学生により課題が選択されると，データベースから抽出された課題テキストが表 3.1 のタグをもとに整形され，提示される．
3. 学生は解答を作成し，JPLAS に提出する．
4. JPLAS は，提出された解答の正誤判定を行い，データベースに解答と評点を保存する．
5. JPLAS は，正誤判定結果を学生にフィードバックする．
6. 学生は，正誤判定結果を元に，必要な場合に解答を作り直し，再提出する．

3.2.2 コード実装上の要点

以上に示したように，JPLAS の各問題形式では，それぞれに固有の正誤判定方法があり，それぞれ保存されるデータも異なる．一方，課題名，ユーザ認証，課題の一覧表示などは，統一した表示と処理が求められる部分がある．そのため，これらのコードの統合を見通しの良い形で行う必要がある．

3.3　JPLAS コード記述ルールの提案

本研究では，JPLAS のコード記述ルールを，*MVC(Model-View-Control)* モデルの枠組みに従うこととする．図 3.2 に，今回採用した MVC モデルの枠組みを示す．

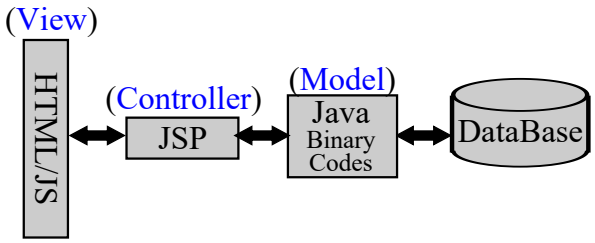

図 3.2: JPLAS に採用した MVC モデル

JPLAS のように多くのユーザインタフェースがメニューのような共通の表示項目をもつ場合，冗長なコードを避けるためには，Ajax の仕組みを利用して，HTML ページの必要な箇所のみ，更新を行えばよい．同時に，サーブレットを用いずに，Ajax を使用することで，JPLAS の開発者は，Tomcat の設定とサーブレットの学習に時間を割く必要がなくなる．

3.3.1　Model のコード記述ルール

図 3.2 に示すように，ここでは，View 部分を HTML および JavaScript，Controller 部分を JSP，Model 部分を Java で実装することとしている．JPLAS の Model 部分では，問題の各ロジックの処理を Java で記述し，そのクラス群で実装する．Model 部分を，View 部分，Control 部分から独立させるために，それに対する入出力は，基本的に HTML タグを持たない普通の文字列，または，その配列で行い，Java 固有のクラスによるデータの受け渡しも行わないこととする．

3.3.1.1　データベース連携機能

JPLAS で採用している MySQL データベースサーバでは，複数のデータベースを有しており，それぞれに複数のテーブルが設定される．JPLAS では，この中の一つのデータベースを使用して，ユーザ，問題，解答の各情報が，それぞれ個別のテーブルに収められている．図 3.3 に，データベース関連のクラスの概念図を示す．

図 3.3 において，左側が JPLAS の使用するデータベース，中央が Java のクラス，右側が各クラスの機能概要である．以下に，各クラスの概要を示す．

(1) データベースクラス

データベースをモデル化するデータベースクラスでは，セキュリティの向上とシステムの変更を容易にするため，データベースとの連携に必要な情報を一元管理する．

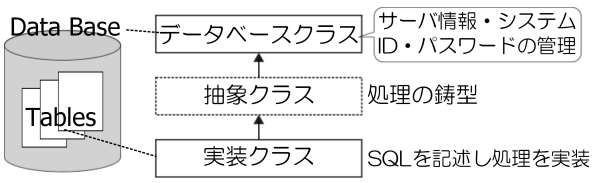

図 3.3: データベース関連クラスの概念図

サーバの IP アドレス，ポート番号，データベース名，システム ID(JPLAS がデータベースサーバに接続する際に使用する ID) とそのパスワードをプライベート変数として隠蔽する．それらのデフォルトの値としては，Tomcat がユーザ認証に使用するデータベース設定を流用する．

(2) 各テーブルクラス

ユーザ，課題，解答の各テーブルをモデル化して各テーブルクラスを作成する．テーブルクラスでは SQL を用いてデータの加工，呼出を行う．また，このクラスは対応するテーブルが存在しない場合に自動的にテーブル生成を行い（初期化メソッド），プログラマがデータベースを直接操作して，そのテーブルの構造を調査せずにすむように項目の定義を保持する．

(3) 抽象クラス

各テーブルでの SQL 呼出しと初期化メソッドは，多くの場合同じ手順で行われる．例えば，データベースの閉め忘れのような手順の抜け防止と，出力の型を統一するために抽象クラスを作成する．各テーブルクラスは，この抽象クラスを実装する形でモデル化する．

データベースは上記の 3 種類のクラスでモデル化され，課題提示機能や課題一覧機能といった，データベースのテーブルを使用する処理では，常に (2) で示したテーブルクラスを使用する．

3.3.1.2 正誤判定機能

今回，問題形式毎に異なる 3 種類の正誤判定機能の中で，学生から提出された解答に適合した機能を自動的に選択するために，**レスポンシビリティ・チェーンデザインパターン** [30] を用いる．レスポンシビリティチェーンでは，図 3.4 のように，データを受け取った処理がそのデータを処理できなかった場合に，予め設定された次の処理へ次々に処理を任せることができる．

この処理を行うために，正誤判定機能をモデル化したクラスでは，与えられたデータが「この処理が可能かどうかを調べる」メソッド，可能であれば「正誤判定を行う」メソッド，可能でない場合の「次の処理を指定する」メソッドが必要となる．これらのメソッドを含んだ各モデルは，それぞれのメソッド名を統一するために，共通の親クラスを持たなければならない．

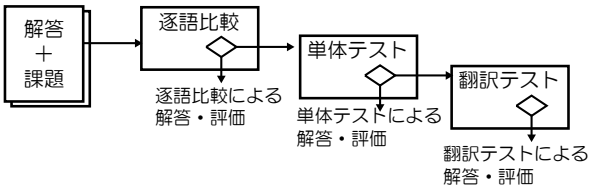

図 3.4: 正誤判定機能のレスポンシビリティチェーン

この要件を満たすために，正誤判定機能の親クラスとなる抽象クラスを作成する．各正誤判定機能のクラスは全て，この抽象クラスの子クラスとして定義される．正誤判定機能のクラスへの入力は，学生の解答およびデータベース内の課題の元データであり，そこからの出力は，メッセージの文字列と評点を表す数値である．

(1) 逐語比較テストによる正誤判定

課題の元データに正解語のリストがあれば，この正誤判定機能を実行する．解答と正解語リストを一語ごとに比較し，正誤判定結果を○（正解）または×（誤答）で表した文字列を返す．また，評点は正答率で表す．次の正誤判定機能として，単体テストを指定する．

(2) 単体テストによる正誤判定

課題の元データにテストコードがあれば，この正誤判定機能を実行する．単体テストによる正誤判定では，通信データとして送られてきた解答とデータベースから呼び出されたテストコードを，実際にサーバのストレージ上にファイルとして書き込み，単体テストのコマンドラインを呼び出すことで行う．

実際の処理は 2 段階で行われる．まず，コンパイルテストを行う．その評点が 100 であれば作業フォルダを作成し，そこに解答のソースコードとテストコードを書き出す．次に，これらのファイルをコンパイルするためのコマンドラインと，さらに単体テストのためのコマンドラインを作る．コンパイル用のコマンドラインを実行し 2 つのバイナリコードが得られたら，単体テスト用のコマンドラインを使用して JUnit による単体テストを実行し，標準出力・エラー出力を文字列として得る．次の正誤判定機能として，コンパイルテストを指定する．

(3) コンパイルテストによる正誤判定

いずれのテストも行われなかった場合，コンパイルテストのみをおこなう．まず，作業フォルダを作成し，そこにソースコードを書き出す．このソースコードのみをコンパイルするためのコマンドラインを作成して実行する．その結果の標準出力・エラー出力を文字列として得る．得られた文字列と，0 点（コンパイル失敗）または 100 点（コンパイル成功）が評点として出力される．コンパイルテストは，単体テストの一部でもあるが，テストコードが準備されていない場合の解答に対する独立したテストとしても使用する．なお，次の正誤判定機能は呼ばれない．

3.3.2 View 部分

View 部分では，*CSS* フレームワークを使用する．CSS フレームワークは Web 標準下で Web ページのコンテンツを表す HTML を，カスケーディングスタイルシート (CSS) を用いて統一性のあるデザインを提供する枠組みである．CSS フレームワークとしては Bootstrap[19] が有名であるが，今回は提案する JPLAS のソフトウェアアーキテクチャが，特定の CSS フレームワークに依存したものではないことを示すため，それ以外の CSS フレームワークとして，SkyBlue[20] を採用する．図 3.5 に，Web ブラウザに表示する画面の構成とサーバとの連携を示す．

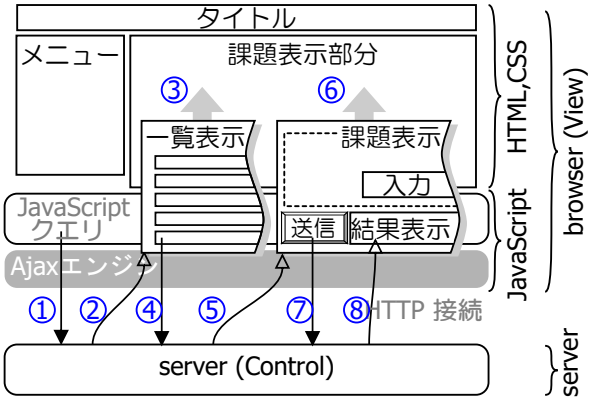

図 3.5: JPLAS の画面構成と Ajax

ブラウザに表示する画面は,「タイトル部分」,「メニュー部分」,「課題表示部分」の 3 つで構成される．JPLAS サーバが学生からのアクセスを受けると，この画面構成の HTML が CSS とともにブラウザに転送される．以下に，画面構成が転送された後の画面の変化を学生が解答する手順に沿って示す．画面の変化は，その構成を保持したまま，変更部分の書き換えのために転送されたデータを，そこに上書きすることで行われる．なお，各転送の要求は，JavaScript による Ajax の枠組みを用いて，Control 部分に対して行われる．[10]

① JavaScript が Control 部分に課題一覧の転送を要求する．

② サーバが課題一覧をテーブル形式で転送する．

③ JavaScript が課題表示部分を更新する．

④ 学生による課題一覧のクリックに応じて JavaScript が Control 部分に課題の転送を要求する．

⑤ サーバが課題に「入力欄」,「解答ボタン」,「結果表示欄」を挿入してから転送する．

⑥ JavaScript が課題表示部分を更新する．

⑦ 学生が解答を入力し，解答ボタンを押すと，JavaScript が解答をまとめて Control 部分に送り，サーバに正誤判定を要求する．サーバで解答の正誤判定を行う．

⑧ JavaScript が正誤判定メッセージを結果表示欄に表示 (上書き表示) する．

⑨ ⑦に戻る

3.3.3 Control 部分

Control 部分は， JSP を用いて実装する．View 部分からのリクエストを受けて，Model 部分を実行し，文字列でその結果を得る．Control 部分では，得られた文字列を HTML として表示するための加工を行う．

1. 課題の一覧表示を行うために（図 3.5 の①で View 部分の JavaScript からアクセスされてサーバで Control 部分が起動し，その出力を②でビューに戻す.），Model 部分から課題の一覧を文字列の 2 次元配列で受け取り，HTML のテーブル要素として View 部分に送る．
2. 選択された課題の表示を行うために（図 3.5 の④から⑤），Model 部分から課題を文字列で受け取り，HTML 上で課題として表示できるように整形し，View 部分に送る．
3. 送られてきた解答の正誤判定を行うために（図 3.5 の⑦から⑧），Model 部分から正誤判定メッセージを受け取り，View 部分に送る．

3.4 提案するソフトウェアアーキテクチャに基づく JPLAS の実装

本章では，提案するソフトウェアアーキテクチャに基づいた，JPLAS の実装について述べる．今回，学生支援機能のみを実装した．教員支援機能の実装は，今後の課題である．

3.4.1 実装の概要

本実装では，CSS で装飾された index.html を使用している．図 3.5 の①は，index.html がサーバからブラウザに転送された後，直ちに実行される JavaScript の関数として実装されている．図 3.5 の④は，一覧表示で学生による課題の選択後， URL を経由して①と同様の処理を行う．図 3.6 に JPLAS の実装を示す．

図 3.6 において丸囲みの数字は，図 3.5 と同じものである．これらは JavaScript で実装されている．Ⓐは課題一覧を作成する JSP ファイル「list.jsp」，Ⓑは課題表示を行う JSP ファイル「question.jsp」，Ⓒは正誤判定機能を呼び出す JSP ファイル「answer.jsp」として実装した．また，白抜きのⒹは，データベース連携機能の実装部分，白抜きのⒺは，解答機能の実装を表す．

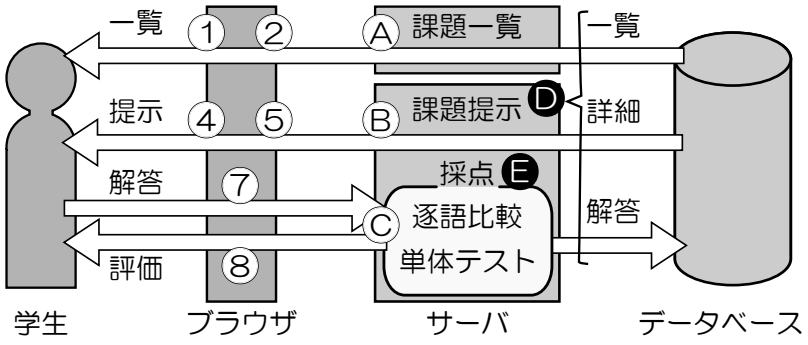

図 3.6: JPLAS の実装の概要

3.4.2 データベース連携機能の実装

図 3.7 に，図 3.6 の白抜きⒹに相当するデータベース連携クラスのクラス図を示す.

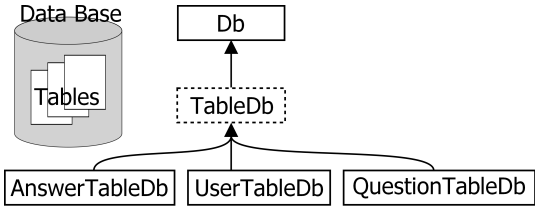

図 3.7: データベース連携機能の実装

図 3.7 において，親となるデータベースクラスは「Db.class」として実装した．各テーブルクラスの鋳型となる抽象クラスは,「TableDb.class」として実装した．各テーブルクラスは，ユーザ情報をもつテーブルを「UserTableDb.class」，課題データをもつテーブルを「QuestionTableDb.class」，学生の解答を蓄積するテーブルを「AnswerTableDb.class」として実装した.

3.4.2.1 問題テーブルとグループテーブル

JPLAS のデータベースでは，登録されたすべての問題は，その問題形式に無関係に，各問題に付与された"ID"を使用して，"問題テーブル"で管理される．その中からいくつかの問題が「課題」としてグループ化され， Java プログラミング授業の履修学生に示される．この問題グループは，"グループテーブル"とタグを用いて管理される．このテーブルのレコードは，各グループのタグと問題 ID のペアで表わされる.

3.4.3 正誤判定機能の実装

図 3.8 に，図 3.6 の白抜きⒺに相当する正誤判定機能のクラスのクラス図を示す.

JPLAS の 3 種類の正誤判定機能は，抽象クラス「Mark」を実装したクラスとして定義した．逐語比較は「BlankMark.class」，翻訳テストは「CompileMark.class」，単体テス

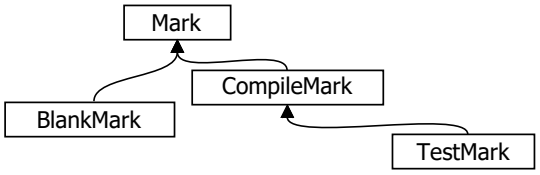

図 3.8: 正誤判定機能用クラスのクラス図

トは「TestMark.class」として実装した.「処理が可能かどうかを判断する」メソッドは「canMark()」,「正誤判定する」メソッドは「mark()」,「次の処理を指定する」メソッドは「setNext()」として実装した．３種類の正誤判定機能は,「setNext()」によるチェーンの順に沿って，自動的に適した方法が取られるように実装した．その際の既定の順序は，逐語比較，単体テスト，コンパイルテストの順とした．

3.4.4　バイナリファイルのアップロード・ダウンロード機能の実装

JPLAS では，ユーザによるオフラインでの解答を可能とするため，課題の全問題の問題文，問題コード，テストコード，正解をダウンロードする機能と，オフラインでの解答結果をサーバにアップロードする機能を提供する．これらのデータファイルは，ダウンロードまたはアップロードする前に，ファイルの圧縮形式を使用して単一のバイナリファイルに圧縮される．

データベースでは，バイナリファイルは，*BLOB* [22] として格納される．そのため，バイナリファイルをダウンロードするには，JSP ファイルのヘッダを調整し配置する必要がある．バイナリファイルのアップロード時，"Commons FileUpload"パッケージが JSP ファイルで使用される． それにより，バイナリファイルが Java クラスのストリームに変換される．[23].

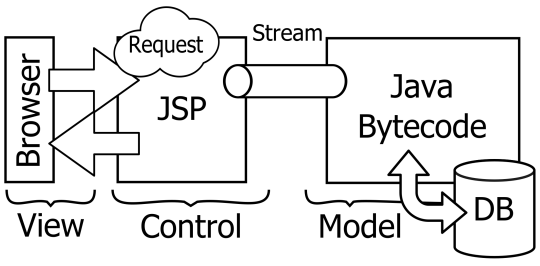

図 3.9: Binary files uploading/downloading.

図 3.9 に示すように，ストリームを使用してバイナリファイルを（ファイルシステムに格納せずに）送信するには，次の手順を適用する．

1) ユーザはアップロードするファイルを，<form> 要素中の <input type = "file"> HTML 要素で選択する ．

2) ファイルは POST メソッドで転送される．

3) JSP ファイルにはファイルの「引き渡し先」が記述されている

4) JSP ファイルは request を受け取り，一つ一つのファイルについて元のファイル名を解析する．

5) JSP ファイルはファイルを"InputStream"オブジェクトとして Java クラスファイルに引き渡す．

コード 3.1: jsp によるバイナリファイルの出力例

```
1 <% @ page contentType="image/jpeg" import="java.io.*,..." %><%
2 InputStream in = ...;
3 response.setContentType("image/jpeg");
4 OutputStream os = response.getOutputStream();
5 IOUtils.copy(in, os);
6 os.flush();
7 %>
```

3.5 評価

提案するソフトウェアアーキテクチャに従って構築した JPLAS の評価のために，まず，今回の実装に必要なファイル数を，従来の実装でのファイル数と比較する．次に，再構築後の JPLAS における， 2 つの新機能追加の事例における追加ファイル数を評価する．

3.5.1 従来の JPLAS とのファイル数比較

表 3.2 に，2 つの JPLAS 実装で使用されているファイル数の比較を示す．ここでは，特にデータベースアクセス関連のコードの改善を示すために，直接データベースにアクセスしているファイル数を比較した．なお，View 部分の HTML, Javascript, CSS の各ファイル，Control 部分の JSP ファイル， Model 部分の Java コードのソースファイルは，それらの拡張子で分類している．ここで，従来の JPLAS では，システムの仕様書を表す JavaDoc と今回の 2 種類の拡張機能は省いた．また，今回の JPLAS では，採用した CSS フレームワーク全体を含めた．提案ソフトウェアアーキテクチャにより，今回の JPLAS 実装では，従来に対して 3 割弱のファイル数で実装されており，データベースアクセス機能も集約されていることがわかる．

3.5.2 JPLAS への新機能追加事例の評価

次に，今回の JPLAS 実装における，2 つの新機能の追加事例での追加ファイル数を評価した．

表 3.2: JPLAS 実装のファイル数比較

ファイル拡張子	従来	今回
java	81	21
データベースアクセス	7	1
jsp	61	12
データベースアクセス	16	0
css	9	6
js	40	38
html	122	11

3.5.2.1 リーダブルコード学習ツールの開発事例

本事例では，コーディング規約に従った可読性の高いコード（リーダブルコード）を書くための学習ツールが実装されている．ここでは，学生が記述したソースコードに対して，命名規則，コーディングスタイル，潜在的問題を，Checkstyle[35], PMD[36] といったツールを用いることで検査し，検査結果を即座にフィードバックする [24].

3.5.2.2 オフライン解答機能の開発事例

開発途上国などでは，インターネットアクセスが保障されないため，オンラインの JPLAS の利用は困難である．ネットワーク回線状況に関わらず JPLAS を利用可能にする必要があることから，本事例ではオフラインのブラウザ上で動作するエレメント補充問題の解答機能が実装されている [25].

3.5.2.3 追加ファイル数

表 3.3 に，今回の JPLAS 実装での 2 つの開発事例における追加ファイル数を示す．表 3.2 と同様に，HTML, Javascript, CSS ファイル，JSP ファイル，Java コードのソースファイル数を拡張子で分類している．

表 3.3: 今回の JPLAS 実装での追加機能ファイル数

ファイル拡張子	リーダブルコード	オフライン解答機能
java	1	0
jsp	10	0
css	17	0
js	31	5
html	3	0

いずれの機能追加でも，データベースアクセスのための新たなファイルは不要であった.「リーダブルコード学習ツール」では，プログラミングコードの表示のために 28 の

JavaScript ファイルと 17 の CSS ファイルが追加された．この比率は大きいが，これは本研究で採用した CSS フレームワークの提供する表示機能では，プログラミング教育支援サイト構築には不充分であったことを示している．いずれの事例においても，従来の JPLAS 実装のファイル数に対して，追加ファイル数をその半分程度に抑えることができている．この理由として，以下が考えられる．

- Java のソースファイルにおいて，View 部分が分離された結果，コードを複製してタグを付け替えただけといった冗長なコードが排除された，
- View 部分においても，CSS フレームワークを利用した結果，より統一的なデザインをより少ないコード数で実現した．
- ユーザの作業毎に発生する Web ページの更新を，Ajax を用いて部分的な更新で行なった結果，重複のない HTML ファイルが実現された．

3.6 まとめ

本章では，Java プログラミング学習支援システム JPLAS の実装を，*MVC* モデルに沿ったものとする，ソフトウェアアーキテクチャの提案とそれに基づく実装を行った．今回の JPLAS 実装を評価として，従来の JPLAS 実装でのファイル数との比較と，2 種類の新機能拡張における追加ファイル数で評価した．今後の課題は，教員支援機能の実装，ユーザインターフェースの改善，異なる新機能拡張での提案アーキテクチャの検証である．

第4章　ステートメント空欄補充問題の提案

本章では，JPLAS でのコード作成学習の最初のステップとして，**ステートメント空欄補充問題**を提案する．

4.1　はじめに

ステートメント空欄補充問題は，Java コード中で最も重要なステートメントである，**コアステートメント**を空欄とし，その補充を要求する問題である．JPLAS で提供されている**エレメント空欄補充問題**や**変数値トレース問題**の学習などを通じた，Java の文法学習を一通り終え，これから，それを用いたコードの作成を学ぶ学生にとって，既存の良質なソースコードを読み，その処理や構造を理解し，模倣することが有効である．本問題では，コアステートメントの補充により，学生が与えられたソースコードを理解することを確認する手段を提供している．

コアステートメントは，ソースコード内で最も重要なものである．しかし，**プログラムスライシング** [26] といった，重要なステートメントを抽出するための解析技法を Java のソースコードに適用するには，計算量が膨大となったり，実装が困難であったりするため，別のアルゴリズムが必要である．

そこで本研究では，Java は C に似た文法を有すること，および，C から拡張された要素を有することに着目して，以下の 2 種類の学習すべき要素を想定し，それぞれでのコアステートメントの抽出アルゴリズムを提案する．

1) 要求仕様を充たすためのメソッド内の要素

2) オブジェクト指向言語の特徴であるクラス間連携の要素

それぞれでのコアステートメントの抽出アルゴリズムを以下のように定める．

1. 1) では，文献 [27] に従い，**プログラム依存グラフ** *PDG* (*Program Dependence Graph*) を利用することで抽出する．
2. 2) では，**ドット演算子**と**引数**に注目してステートメントをランキングすることで抽出する [10]．

提案アルゴリズムの評価として，まず，ステートメント補充問題による学習効果を確認するため，本学工学部通信ネットワーク工学科 2 年生を対象に, 実際に本機能を使ってもらった. その結果, 類似した問題について解答速度が向上していた.

次に，5 つの Java コードに適用して抽出されたコアステートメントを，Java の学習者による主観的な抽出結果と比較した．その結果，両者に高い一致がみられ，提案アルゴリズムの有効性が確認された.

本章では，まず，ステートメント補充問題の提案を行う．次に，コアステートメント抽出アルゴリズムの提案を行う．その後，各提案の検証を行う．最後に，本章のまとめと今後の課題を述べる.

4.2　ステートメント補充問題

本章では，ステートメント補充問題の提案を行う.

4.2.0.4　ステートメント補充問題

ステートメント補充問題では，問題コードの中のコアステートメントを空欄とし，学生には，与えられた要件を充たすステートメントの入力が求める．図 4.1 に本問題の例を示す.

ここで，従来のエレメント空欄補充問題と異なり，正解となるステートメントは，通常，一意には記述できないため，テストコードを用いたテストにより，正誤判定を行う．そのため，ブラウザ上で，解答ステートメントを問題コードの空欄部に埋め込み，コードを完成させてから，サーバに送信している.

4.2.1　プログラム依存グラフ

本研究では，コアステートメントの抽出に，**プログラム依存グラフ** *PDG*（*Program Dependency Graph*）を利用する．PDG では，ステートメント間の制御依存関係とデータ依存関係に着目している．まず，ステートメント $s1$ とステートメント $s2$ に対し，以下の場合に， $s1$ から $s2$ への制御依存関係が成り立つものとする.

1. $s1$ は条件文かループである
2. $s2$ が実行されるかどうかは $s1$ に依存する

また，以下の場合に， $s1$ から $s2$ へのデータ依存関係が成り立つものとする.

1. $s1$ において変数 v が定義されている
2. $s1$ から $s2$ への実行可能パス内に v は再定義されない
3. $s2$ において変数 v が参照されている

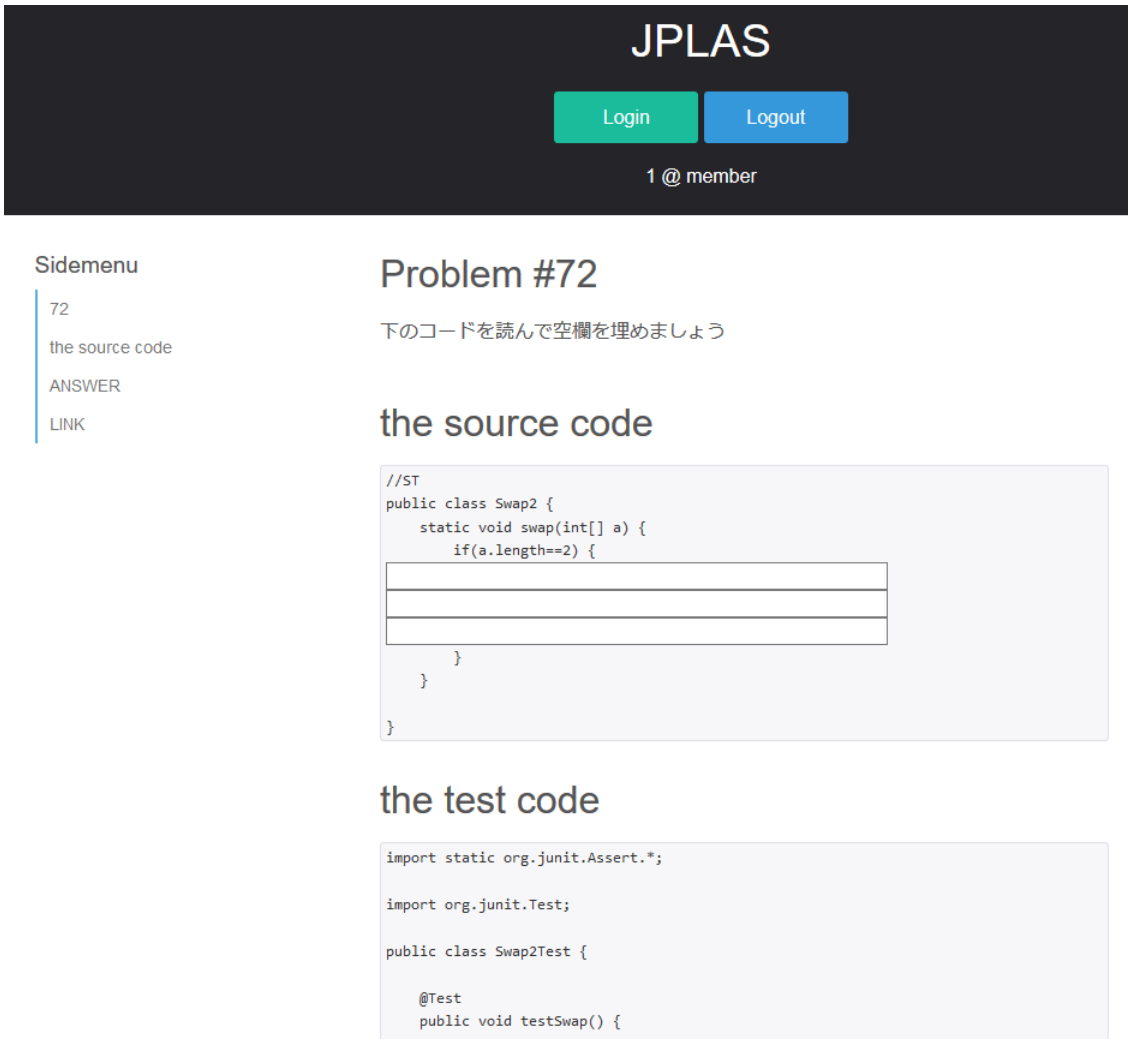

図 4.1: ステートメント補充問題の例

PDG では，各ステートメントが点，各依存関係が有向辺で示される．そして，各ステートメントの重要度（得点）は，この有向辺の入出数（次数）で決められる．

本研究では,「変数」間のデータ依存関係（図 4.2）に加え（[27]）,「オブジェクト」間の依存関係も考慮している．

1. 「変数」には，プリミティブなものに加え,「オブジェクト」も含むこととした．その際，オブジェクト内の変数（メンバ変数）に値の変更などの何らかの操作があった場合，オブジェクトそのものが再定義・参照されたものと見なした．これは同じ操作を，オブジェクトごととメンバ変数ごとに重複してカウントを防ぐためである．

2. オブジェクトへの代入は，代入文の左辺にそのオブジェクトが現れる場合と，オブジェクトのメソッドを呼び出す場合とした．

3. オブジェクトの参照は，代入文の右辺にオブジェクトが現れる場合と，メソッドの引数として使用されている場合とした．

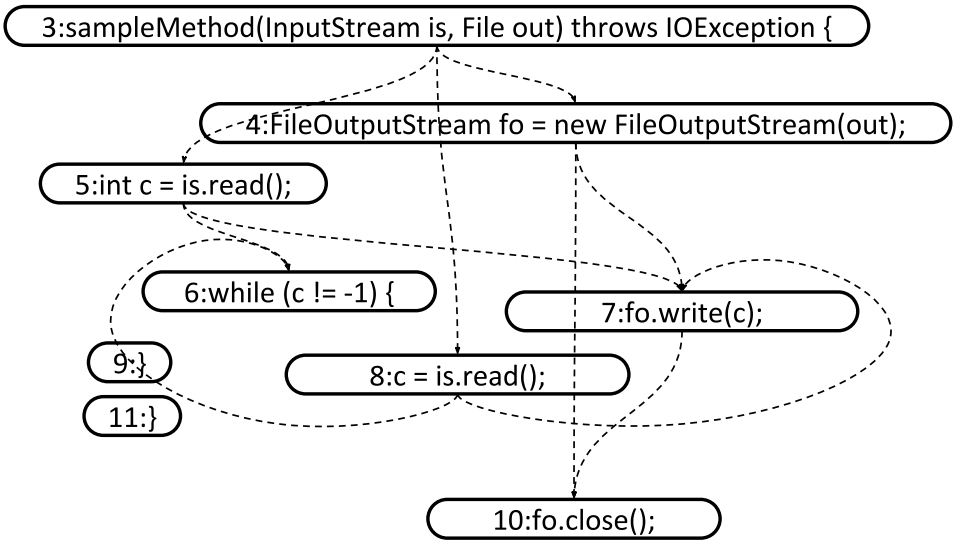

図 4.2: プログラム依存グラフの例

以下に，その例を示す．

「オブジェクト」への拡張例

```
1  b.setTime(a.end.selTime()+i*7*1000*60*60*24);
2  c.next.start = d.parent.start;
```

1 行目の例では，変数 i に加えて a，b も変数と見なす．i は参照，b は再定義， a は参照でありさらに（再）定義でもある．end　は a のメンバ変数であるが，オブジェクトの一部として変数には含まない．2 行目の例では，c，d を変数とし，c が再定義であり，d がは参照である．

4.2.2　プログラム依存グラフでの問題点

PDG を用いてコアステートメントを抽出する際に，コードにおける，1 つの PDG の適用範囲と，文字列定数の扱いに考慮する必要がある．

まず，1 つの PDG は，メソッド（関数）単位で適用する．変数名の再使用，引数，広域変数を考慮したとしても，パッケージ外のクラスやメソッドなどのバイナリファイルで提供されているコードでは，ソースコードからの解析は難しくなることから，1 つの PDG はメソッド単位に限ることとしている．

また，PDG では，文字列定数が抽出されないため，出題者の意図と異なるコアステートメントが抽出される場合がある．以下にその例を示す．

コード 4.1: 出題者の意図と異なるコアステートメント抽出

```
1  public class Sx05 {
2      public static boolean isNumberOnly(final String target) {
```

```
3               if (target!=null && target.length()>0) {
4                       return 0 == target.replaceAll("[0-9]*", "").length();
5               } else {
6                       return false;
7               }
8       }
9   }
```

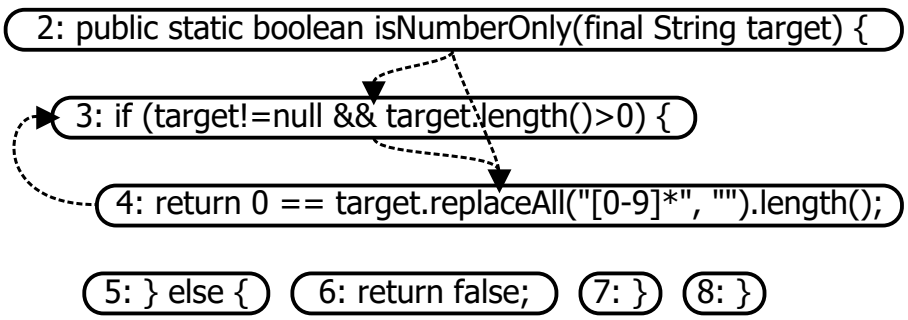

図 4.3: 意図しないステートメントが選択される PDG

図 4.3 の例では，与えられた文字列が数字のみか否かを判定するメソッドが定義されているが，3 行目と 4 行目が同じ次数となり，最も重要となる replaceAll メソッドの戻り値に対して長さを求める 4 行目のステートメントは選択されない．

コード 4.2: クラスメソッドの関数風の記述

```
4   return 0 == strlen(replaceAll("[0-9]*", "",target));
```

ここで，この 4 行目は，C の関数であれば，コード 4.2 のように記述され，学習価値のあるものであるが，本研究では，変数に着目して抽出を行うため，それらを含むステートメントが必ずしも抽出されるわけではない．これらの問題点の対策が必要である．

4.3 コアステートメント抽出アルゴリズムの提案

本章では，コアステートメント抽出アルゴリズムの提案を行う．ここでは，前節で示した問題点を解決するために，Java プログラミングにおいて学ぶべき要素を，以下の 2 種類とし，それぞれでのコアステートメント抽出アルゴリズムを提案する．

1) 要求仕様を充たすための**メソッド内要素**

2) オブジェクト指向言語の特徴である**クラス間連携要素**

メソッド内要素は，Java 言語のメソッド内部に相当する，そのコアステートメントは，PDG を用いて抽出する．

クラス間連携要素は，オブジェクト指向の特徴であるクラス間連携に関連する部分であり，PDG を用いて，そのコアステートメントを抽出することが出来ない．そのため，本

研究では，クラス間連携を利用しているステートメントをコアステートメントとして抽出する．

4.3.1 メソッド内要素のコアステートメント抽出

メソッド内要素のコアステートメントは，各メソッドを対象に PDG を利用して抽出する．PDG では， 図 4.2 のようにステートメント間のデータ依存関係のみに注目する．以下に本手続きを示す．

1. 読み込んだソースコードからコメント部分を削除し，作問用のソースコードとする．
2. 作問用のソースコードをセミコロン（";"）と左波括弧（"{ "）を手がかりにステートメント単位に分割する．
3. 各ステートメントをエレメント単位に分割する．
4. 以下の条件を満たすエレメントを変数として抽出する．
   (a) 先頭の文字がアルファベットである．
   (b) 予約語，クラスでない．
   (c) メソッドではない．これは，その後に右括弧（ ”(“ ）が続かないことで判断する．
5. 変数の中で，ドット演算子が前置されないものを処理対象として選出する．ここで，ドット演算子が前置された変数は，インスタンス化されたオブジェクト内部の変数のため，除外する．
6. 各変数を以下のルールで**定義変数**または**参照変数**に分類する．
   (a) 定義句（直前の語が型指定）があれば定義変数
   (b) 等号の左辺であれば定義変数
   (c) 等号の右辺であれば参照変数
   (d) メソッドが呼ばれていれば定義変数
   (e) 引数は参照変数
7. 定義変数から参照変数への辺を生成する．
8. 辺の数をステートメント毎に集計する．

4.3.2　クラス間連携要素のコアステートメント抽出

クラス間連携要素のコアステートメントの抽出では，**ドット演算子**と**引数**に注目する．以下にその手続きを示す．

1. 指定されたソースコードと同じフォルダにあるソースファイルを読み込む．
2. 読み込んだソースコードをセミコロン（";”）と左波括弧（”{”）を手がかりにステートメントに分割する．
3. ステートメントを単語に分割する．なお，ここではコメントや文字列定数は 1 語として取り扱う．
4. 各ステートメントで以下の 2 つの値を算出する．
   (a) ドット演算子によって連結されたエレメントの個数
   (b) 入れ子を含む引数の数
5. ただし，アルゴリズムやプログラムの機能とは関連の薄いドット演算子の影響を防ぐために，以下のドット演算子については対象から除外する．
   (a) package 文と import 文に含まれるものは，C において include 文でヘッダファイルの位置を示すのと同様にクラスの位置情報を示す役割であるため，本研究の対象から除外する．
   (b) System.out と例外処理の e.printStackTrace は定型句となっているため，対象から除外する．

4.3.3　抽出ステートメントの間引き

提案アルゴリズムを用いて，連続するステートメントが抽出された場合，問題が難しくなりすぎるのを防ぐため，最初のステートメントのみを選択した．

4.3.4　抽出結果の例

以上のアルゴリズムを用いて抽出したコアステートメントの例を以下に示す．

4.4　メソッド内要素のみでの提案法の評価

本章では，4.3 章で示したメソッド内要素のみに対する提案アルゴリズムおよびそれによるステートメント空欄補充問題の有効性を検証する，そのため，本学科の Java プログラミング授業の受講者に本機能を適用し，その評価結果を示す．

表 4.1: メソッド内要素に対するコアステートメントの抽出例

pdg	行番号	コード
0	1:	public class Sx05{
0	2:	public static boolean isNumberOnly(String target){
5	3:	if(target != null && target.length() > 0){
4	4:	return 00 == target.replaceAll("[0-9]*","").length();
0	5:	else
0	6:	return false;
0	7:	}
0	8:	}
0	9:	}

4.4.1 評価対象者と課題

本授業の受講者は，本学科の 2 年生が中心であり，C プログラミングおよび C++プログラミングについて，それぞれ半年間学んだ後，Java プログラミングを学んでいる．本評価のための課題として，提案機能を用いて，ステートメント補充問題の課題を 27 問作成した．その内訳は，main メソッドでの標準入出力の使用を中心としたコードから作成した 16 問と，IT 企業から提供された，入力データをチェックするためのスタティックなメソッドを中心としたコードから作成した 11 問とした．これらの課題を，11 月中旬より 4 週間自習させた後，授業時間を用いて小テストを実施した．

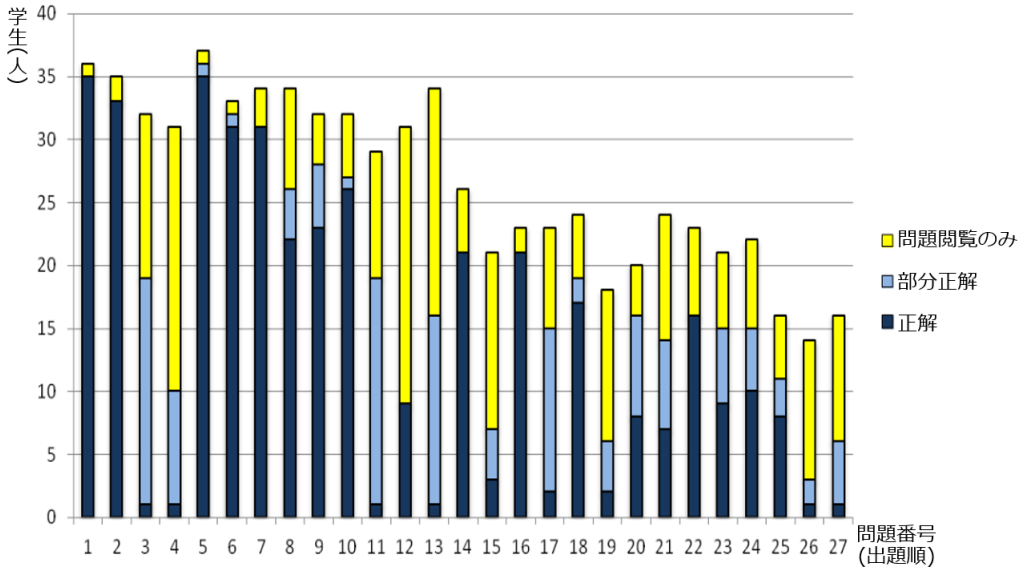

図 4.4: 課題解答数

表 4.2: クラス間連携要素に対するコアステートメントの抽出例

dot	para.	行番号	コード
0	0	1:	public class Main{
0	0	2:	public static void main(String[] args){
0	1	3:	Support alice=new NoSupport("Alice");
0	2	4:	Support bob=new LimitSupport("Bob",100);
0	2	5:	Support charlie=new SpecialSupport("Charlie",429);
0	2	6:	Support diana=new LimitSupport("Diana",200);
0	1	7:	Support elmo=new OddSupport("Elmo",100);
0	2	8:	Support fred=new LimitSupport("Fred",300);
5	5	9:	alice.setNext(bob).setNext(charlie)
			.setNext(diana).setNext(elmo).setNext(fred);
0	0	10:	for(int i = 0; i < 500; i+=33){
1	2	11:	alice.support(new Trouble(i));
0	0	12:	}
0	0	13:	}
0	0	14:	}

4.4.2 課題解答数

まず，図 4.4 に，課題毎の学生の解答数を示す．ここでは，縦軸が学生数 (人)， 横軸が課題番号を示している．各グラフでは，テストコードの全項目をパスする解答を提出したものを comp，テストコードの一部がクリアできなかったものを mid，問題の参照のみで回答に至らなかったもの，コンパイル時にエラーが出たままのもの，テストコードを全くパスできなかったものを zero として示している．

4.4.2.1 課題解答時間

次に，図 4.5 に，各課題について，問題を初めて見た時刻から正解となる解答が提出できた時刻までの経過時間（解答に要した時間）を示す．課題は，通常，その課題番号の昇順に解かれるため，演習時間の経過と共に，学生が解答を作成するのに必要な時間は減少している．すなわち，本機能を用いた演習を行うことで，コードリーディングの力が増したと言える．

4.4.3 類似課題の解答時間

類似の処理を扱うコードに接する機会が多いほど，学生にとって容易なものになり，解答に要する時間は短くなると思われる．そのため，全課題の中から類似の処理を扱う課題を拾い出して，解析を行った．

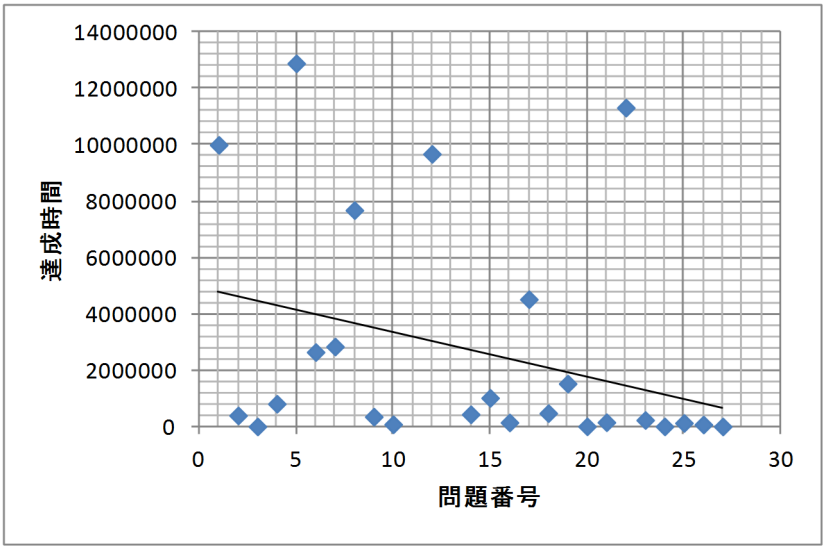

図 4.5: 課題解答時間

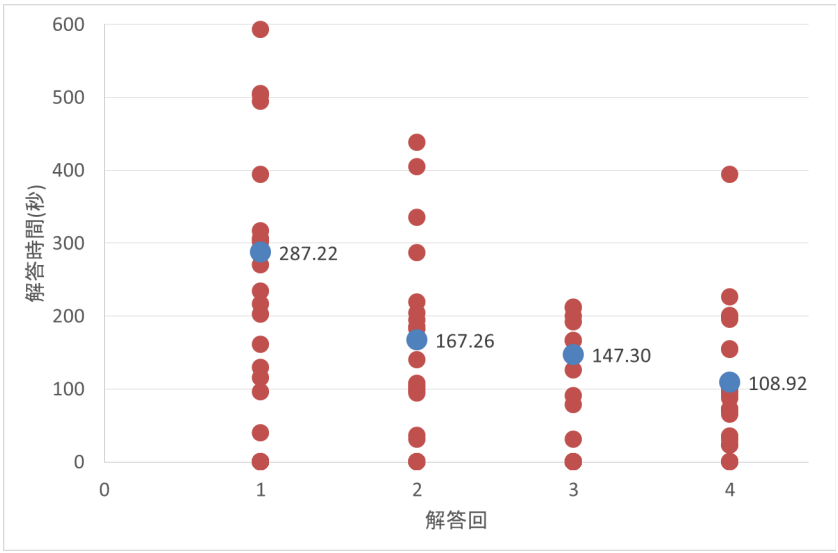

図 4.6: readLine 課題での解答時間

まず，27 課題中，4 課題で，バッファから行単位に呼び出す readLine を出題している．図 4.6 に，これらの課題を学生毎に，解いた順に並び替え，課題に要した時間を示す．その際，課題に要する時間中には退席するなどの時間も含まれているため，その影響を排除するために 10 分以内の解答時間に限定した．図 4.6 より，平均値で 1 回目 287 秒， 2 回目 167 秒， 3 回目 147 秒， 4 回目 108 秒と，解答の回数を重ねるごとに課題に要する時間が短くなっていることが判る．

次に，27 課題中，4 課題で，引数として指定される文字列クラスが有効かどうかを判定するコードが出題されている．図 4.7 に，これらの課題に要した時間を示す．今回も 10 分以内の解答時間に限定した．図 4.7 より，平均値で 1 回目 287 秒， 2 回目 167 秒と，解答の回数を重ねるごとに課題に要する時間が短くなっていることが判る．これらの結果

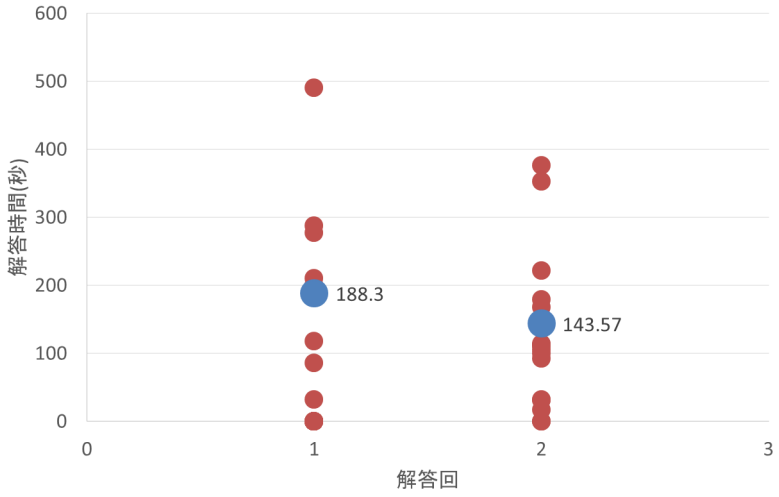

図 4.7: 文字列検査課題での解答時間

からも，提案するステートメント補充問題はコードリーディング力の育成に有効であると言える．

4.5　全要素での提案法の検証

本章では，4.3 章で示した 3 種類の要素すべてに対する提案アルゴリズム，および，それによるステートメント空欄補充問題の有効性を検証する，そのため，本学科の学生 50 名に本問題を適用し，その評価結果を示す．

4.5.1　検証に用いた Java コード

本検証では，3 種類の要素すべてを含む，以下の 5 つの Java ソースコードを使用した．それぞれ，複数のクラスを持つ，ステートメント数 24，45，54，32，40 のプログラムである．package 毎に，1 つのソースファイルで作成したもの，複数のソースファイルに分けたものを取り混ぜている．これらのコードに，提案アルゴリズムを適用し，コアステートメントを抽出した．

1. Java アプリケーションの「フレーム」の実装コード（Window） [31]
2. 「バイナリサーチアルゴリズム」の実装コード（BinarySearch） [32]
3. デザインパターンの「プロトタイプ」の実装コード（Prototype） [30]
4. デザインパターンの「シングルトン」の実装コード（Singleton） [30]
5. デザインパターンの「ビルダー」の実装コード（Builder） [30]

4.5.2 比較対象

次に，学生にこれらのソースコードを提示し，各自でステートメント空欄補充問題を作成する場合，どの行を選択するかを選んで貰った．その際，複数ステートメントの選択も可とし，その場合には，順位を振って貰うこととした．学生は，本研究室の学生 7 名，本学科一般学生 43 名である．前者は，Java を用いた研究に従事しており，その基本には熟知しているが，熟練度にはばらつきがある．後者は，2 年生で，C と C++の履修後，現在，Java プログラミングの授業を履修中である．

4.5.3 Java プログラミング授業の進め方

ここで，対象学生（学習者）が本調査を行うまでの Java プログラミングの授業の進め方を紹介する．まず，最初の 4 回の授業では，C の文法事項の復習から Java への導入に行った．ここでの演習課題には，JPLAS のエレメント空欄補充問題を与えた．

次の 4 回では，String クラスのメソッドに出現する正規表現やデータベース連携のための SQL を中心に，ストリームや Window アプリケーションの作成を学習した．その際，Java 豊富なライブラリを使用するためのマニュアルである，Javadoc[29] の読み方を紹介した．ここでの演習課題には，JPLAS の 3 つの問題，すなわち，エレメント空欄補充問題，ステートメント空欄補充問題，コード作成問題のすべてを与えた．

最後の 4 回では，洗練されたオブジェクト指向プログラムの例として，*Gof* のデザインパターン [30] を取り上げた．ここでの演習課題には，学生個々に，自由なテーマでのアプリケーションを作成させた．なお，新規の JPLAS の演習課題は課さず，解答の遅れた学生の挽回期とした．

4.5.4 アルゴリズムによるコアステートメント抽出結果

各 Java コードの 3 つの要素における，提案アルゴリズムによるコアステートメント抽出数を，表 4.3 に示す．ここで，メソッド内要素のコアステートメント抽出では，メソッド毎の PDG を用いて 1 つのステートメントのみを選択することから，その数はメソッド数に等しくなる．

4.5.5 学習者によるコアステートメント抽出結果

学習者によるコアステートメント抽出では，全学生から総数で 351 のステートメントが回答され，その中から重複を排除すると，ステートメント数は 93 であった．その中には，アノテーション（@Override など）のみのステートメントなど，今回，提案アルゴリズムで対象としていない要素が含まれており，学習者の抽出結果から除外することとした．また，少数の学習者のみの回答を例外として除くために，ステートメント毎に平均回答数を上回るもののみに限定した．これらにより，21 ステートメントが残った．

表 4.3: コアステートメント抽出数

コード	クラス数	メソッド数	行数	クラス内要素	クラス間連携	外部連携	学習者
Window	2	2	24	2	3	0	2
BinarySearch	2	3	45	3	3	0	3
Prototype	4	6	54	6	5	0	2
Singleton	2	3	32	2	3	0	3
Builder	4	6	40	6	6	0	5

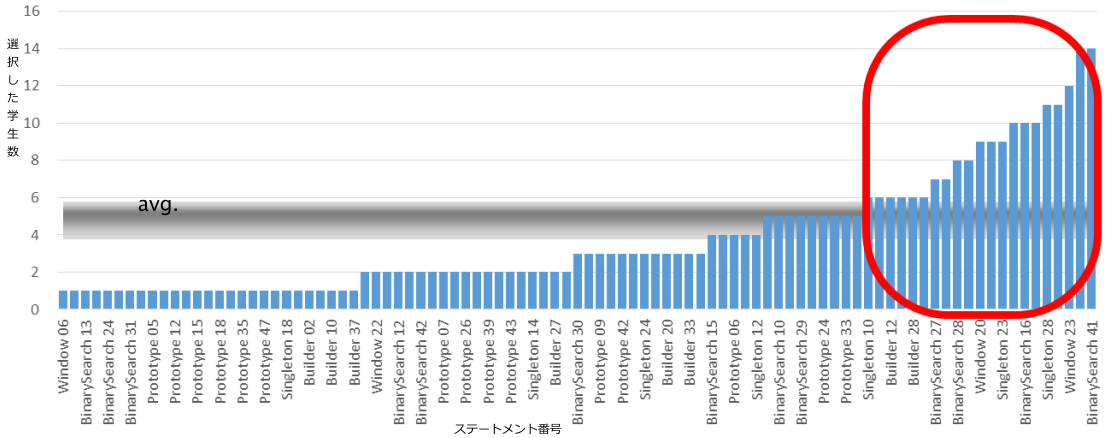

図 4.8: 5 つの Java コードに対するコアステートメント抽出数の比較

更に，今回の対象外である，Java の文法やライブラリが問題となっている 6 つのステートメントを除外した．その結果，学習者によるコアステートメント抽出結果として，15 ステートメントとなった．

ここで，最後に除外した 6 ステートメントは，以下のコード例に示すように，いずれも，new を含んでおり，new 以外には配列，Map，HashMap，ArrayList などを含んでいる．なお，new を含むステートメントの数は，全コードで 15 である．

コード 4.3: 最後に除外した 6 ステートメント

```
1  WindowTest windowText = new WindowTest();
2  frame = new Frame("Window Test"); // Set the initial value for the Frame
3  Entry[] table = new Entry[MAX]; // Array to store the data
4  private Map <String,  Prototype> hashmap = new HashMap <String ,
       Prototype> ();
5  List list = new ArrayList();
6  Director director = new Director(new Con...);
```

以上で得られた，学習者のコアステートメント抽出数を，表 4.3 の「学習者」の欄に示す．ここでは，学習者は，2 種類の要素の区別を意識せずに抽出しているため，その総数

としている．

4.5.6　アルゴリズムと学習者抽出の比較

5 つの Java コードに対する，提案アルゴリズム，および，学習者によるコアステートメント抽出結果を比較する．ここでは，最もオブジェクト指向言語としての重要な要素である，クラス間連携要素のみとして比較した場合と，2 つの要素すべてで比較した場合について議論する．

1. クラス間連携要素のみでの比較

クラス間連携要素のみを考慮した場合，提案アルゴリズムによるコアステートメントの抽出数は，5 つの Java コード全体で 20 である．これに対し，学習者による抽出は 15 である．これらの中で，同じステートメントを選択しているものは 12 である．このことから，クラス間連携要素のみでの比較では，学習者の抽出したステートメントの中で，80% を提案アルゴリズムでも抽出可能であると言える．

クラス間連携要素のみに限定した場合の提案アルゴリズム，学習者によるコアステートメント抽出結果の比較を，表 4.4 に示す．ここで,「重複」は両手法で共通のステートメント数,「提案」は提案アルゴリズムのステートメント数,「学習者」は学習者の抽出したステートメント数,「行数」は各 Java コードの総ステートメント数,「重複率」は重複ステートメント数を学習者の抽出したステートメント数で除した値である．

表 4.4: クラス間連携要素のみでのコアステートメント抽出結果の比較

コード	重複	提案	学習者	行数
Window	2	3	2	24
BinarySearch	1	3	3	45
Prototype	2	5	2	54
Singleton	2	3	3	32
Builder	5	6	5	40

表 4.4 の結果を，そのステートメント数に関するスケールがわかるようにグラフ化したものを，図 4.8 に示す．本グラフの横軸は，5 つの Java コードを示す．棒グラフの各部分は，それぞれの Java コードにおいて，重複，提案のみ，学習者のみ，それ以外のステートメント数の割合を示す．また，折れ線グラフは，学習者の抽出したステートメントを提案アルゴリズムが抽出できた割合を示す．この結果より，5 つの Java コードに対して，提案アルゴリズムでは，その 10.8% のステートメントを選択し，学習者による主観的抽出結果の 80% をカバーすることができたと言える．

2. 2 要素での比較

次に，提案アルゴリズムによるコアステートメントの抽出数は，その 2 要素全体を考慮し

た場合，24 である．これは，全ステートメントの 13.9% となる．しかし，学習者が抽出したステートメントに対する一致率は 80% にとどまった．

この場合の提案アルゴリズム，学習者それぞれによるコアステートメント抽出結果の比較を，表 4.5 に示す．ここで,「提案」における括弧内の数は，他の要素を考慮したことで増加したコアステートメント数である．

表 4.5: コアステートメント抽出数比較 (2 要素)

コード	重複	提案	学習者	行数
Window	2	4(+1)	2	24
BinarySearch	1	4(+1)	3	45
Prototype	2	6(+1)	2	54
Singleton	2	4(+1)	3	32
Builder	5	6(+0)	5	40

この場合，5 つの Java コードの中で，BinarySearch と Singleton において，重複が少なく，個々の抽出数が多いことから，両者の差が大きいと言える．この理由を分析するため，以下の BinarySearch のコードを調査した．

コード 4.4: BinarySearch のコード

```
1  int middle = (low + high) / 2;
2  if (key == table [ middle ]. key) {
3  return table [ middle ]. data; // found
```

このソースコードで学習者が抽出したステートメントは，2 行目と 3 行目である．これに対して提案アルゴリズムによる抽出では，1 行目のステートメントが選ばれている．提案アルゴリズムでは連続したステートメントが抽出された場合，その一行目を選択する．一方，この 3 つのステートメントではいずれが空欄化されても学習効果は期待できる．そのため，提案アルゴリズムを再考し，連続したコアステートメントからの抽出は必ず一行目を選択するのではなく，任意のステートメントを選択するよう変更する．

4.6 まとめ

本研究では，Java プログラミング学習のためのステートメント空欄補充問題において，空欄とするコアステートメントの抽出アルゴリズムを提案した．Java 学習で学ぶべき 2 つの要素を定義し，それぞれでのコアステートメント抽出手順を明らかにした．提案アルゴリズムで生成した問題に関する評価結果より，メソッド内要素に対する，提案アルゴリズムのステートメント補充問題は，コードリーディング力の育成に有効であると言える．また，提案アルゴリズム全体としては Java プログラミングの学習者による抽出結果と比較し，80% 以上の一致を見た．

今回，学習者の抽出したコアステートメントには，アノテーションや Java 特有の文法，ライブラリなどが含まれていた．これらは，提案アルゴリズムで除外しているが，今後は，それらの要素も，コアステートメントとして抽出する必要がある．また，複数クラスを有するコードは長い上に，ファイルも一つとは限らないため，JPLAS の Web ブラウザで表示をするには，UML を活用するなどの工夫が必要と言える．

第5章　コードクローン除去問題の提案

本章では，JPLAS でのコードリファクタリング学習を目的として，**コードクローン除去問題**を提案する．

5.1　はじめに

ソースコードが所要の機能を充足していても, 拡張性, 保守性に難がある場合, 「良い」コードとは言えない. 特に, 安易なコピーペーストや, コード作成に対する未熟な理解から発生する冗長なコード断片は, 保守の阻害要因となり, コード品質の劣化を招く．コード中の互いに一致する, あるいは, 類似したコード断片は, **コードクローン**と呼ばれ, 多くの場合, コピーペーストにより発生している [44]. コードクローンを有するコードは, リファクタリングを考慮すべき, コードの悪い状態であるとされる．リファクタリングは, 「ソフトウェアの外部的振る舞いを保ちつつ内部の構造を改善すること」をいい, 本来は経年開発によるコードの品質低下を防ぐための方法である [44].

JPLAS を含む, プログラミング課題に対する学生による解答コードの作成では, 多くの場合, 出来上がったコードの品質には気を留めず, コンパイルに成功, あるいは, 実行結果が正しければ, そのまま提出し, 次の課題へ移ることが多い．ソフトウェアシステムのリファクタリングに相当する「自らのコードを見直す行為」を怠るため, 指示された機能は充足しているものの, コードクローンを含む, 冗長で低品質の解答コードを提出するといった問題がある.

以上の問題の対策のために, 本章では, JPLAS の新しい問題として, 与えられた Java コード中に含まれるコードクローンを一か所にまとめることで, それらの除去を行う, **コードクローン除去問題**を提案する.

コードクローン除去問題では，**問題コード**として，**コードクローン**（ソースコード中の互いに一致または類似した部分）を含むコードを，**テストコード**とともに提示する．学生はこれを，コードクローンを含まないコードに修正して，**解答コード**として提出する．解答コードは，テストコードを用いた単体テストによる処理の正しさの採点と，コードクローン検出判定による内部構造の改善の採点を行う．

本問題では，コードクローン除去方法を,（１）分岐文や反復文中で行を入れ替える使用文法の適正化,（２）同一クラス内でメソッドを作成するメソッド生成,（３）他のクラスを考慮するクラス生成,（４）高度なクラスの機能を使ったテンプレートメソッド使用, の 4 種類に区別し，それぞれに対応した課題を準備することとしている．また,（２）以降

の問題に対しては，一般に初学者にはその理解が難しいことから，以下の 3 段階での学習法を提案する．

1. 解法の理解
2. 問題コード中のコードクローンの指摘
3. 解答コードの作成

コードクローン除去問題では, 課題毎に, プログラムの仕様を表す「課題文」, コードクローンを含む「ソースコード」, その正しさを検証するための「テストコード」を学生に提示する. 学生は, リファクタリング後のコードを「解答コード」として提出する. サーバにおける解答コードの採点では, コンパイルによる**文法チェック**, JUnit 上でテストコードを用いた**単体テスト**, および, スタティック解析ツール PMD の Copy/Paste Detector (CPD) [36] を用いた**コードクローン検査**を行う.

本章では, まず, JPLAS のコード作成問題の問題点について述べる．次に，コードクローン除去のためのリファクタリング技法の検討を行う. 次に, コードクローン除去問題をリファクタリング技法と関連付けて提案する. その後, 生成したコードクローン除去問題の例題を学生に解答してもらい, 難易度の評価を行う. 最後に，本章のまとめと今後の課題を述べる.

5.2 コード作成問題での問題点

JPLAS の問題を学習することで, 学生は, Java の文法から始め, 徐々に継続的に, プログラミング能力を高めることが期待されている. そして最終的に, コード作成問題を解答することで, 要求される機能を満足するコードを作成することが可能となる. しかしながら, 以下に示す例のように, 冗長で無駄の多い解答コードを作成しても, それを修正して高品質なコードに仕上げる機会が提供されていない.

コード 5.1: 好ましくない正解解答

```
1  class FizBuz{
2      public static void main(String[] args){
3          int n = 1;
4          while(n <= 100){
5              if(n % 15 == 0){
6                  System.out.println("FizBuz");
7                  n = n + 1;
8              }else if(n % 5 == 0){
9                  System.out.println("Buz");
10                 n = n + 1;
11             }else if(n % 3 == 0){
12                 System.out.println("Fiz");
13                 n = n + 1;
14             }else {
```

```
15                          System.out.println(""+n);
16                          n = n + 1;
17                      }
18                  }
19              }
20  }
```

上記のコードは, よく知られた FizzBuzz 問題 [34] に対して, ある学生が実際に提出したコードである. 下線で示した "n = n + 1 ;" のステートメントは, 7 行目, 10 行目, 13 行目, 16 行目に現れるが, 所要の機能を満足しているため, JPLAS では正解としている. しかし, このように反復して現れるコード断片はコードクローンであり, 1 つにまとめるべきである.

5.3 4 種類のコードクローン除去問題

本章では, コードクローン除去のためのリファクタリング技法の違いによって，4 種類のコードクローン除去問題の提案を行う.

5.3.1 コードクローンの定義

一般にコードクローンは, プログラムコード中に現れる同一, あるいは, 類似のコード断片を指し, 主として, コピーペーストで作成される. また，コードクローンは, 通常, 20%程度は存在するといった研究結果 [45] があることから, コードクローン除去問題においても, 全てのコードクローンを除去の対象とすべきではないと言える.

5.3.2 コードクローン除去問題の種類

コードクローンの除去のためのリファクタリング技法として，文献 [44] などを参考に，異なるコードクローン除去手法を有する, 以下の 4 種類の問題を対象とする.

- 使用する文法の適正化によるコードクローン除去に関する問題
- メソッド作成によるコードクローン除去に関する問題
- クラス作成によるコードクローン除去に関する問題
- テンプレートメソッドによるコードクローン除去に関する問題

5.3.3　文法適正化によるコードクローン除去の問題

本問題では, コードクローンの中で, 先に挙げた FizzBuzz のプログラムのように, 使用する文法の不適切さが原因で生じる, 単純な冗長を除去することを目的としている. このようなコードクローンは, 文法の理解が不十分であったり, 処理フローの設計を行わずに行き当たりばったり的にコードを作成した場合に発生する. プログラミング初心者に頻繁に見られるコードクローンである. 先の例では, 7,10,13,16 行目の”n=n+1;” を排し, while 文の最後 (17 行目と 18 行目の間) に挿入する構造に変更するか, for 文を使用して文法的に一か所にまとめることで, コードクローンの除去が可能である.

5.3.4　メソッド作成によるコードクローン除去の問題

本問題では, 同じクラス内で, 繰り返し現れる類似の処理（コードクーロン）を記述するメソッドの作成を行うことで除去を行う. 類似の処理を記述するメソッド（関数）を準備できることが目標である. コード 5.2 に問題コードの例を挙げる.

コード 5.2: メソッド作成によるコードクローン除去の問題例

```
1  public class Animal {
2      void howl(){
3          for(int i = 0; i < 3; i++){
4              System.out.println("Bow−wow");
5          }
6          for(int i = 0; i < 3; i++){
7              System.out.println("Meowing");
8          }
9      }
10 }
```

本問題コードでは, 3 行目から 5 行目, 6 行目から 8 行目までがコードクローンである. 解答コードでは, 文字列変数を引数にもつメソッドを同じクラス内に作成し, 呼び出すことが期待される. 解答コードの例をコード 5.3 に示す.

コード 5.3: メソッド作成によるコードクローン除去の問題解答例

```
1  public class Animal {
2      void howl(){
3          howlPart("Bow-wow");
4          howlPart("Meowing");
5      }
6      void howlPart(String howlingStr){
7          for(int i = 0; i < 3; i++){
8              System.out.println(howlingStr);
9          }
10     }
11 }
```

5.3.5　クラス作成によるコードクローン除去の問題

本問題では, 同じコードクローンを持ったクラスが共通で利用できるクラスを定義することで, コードクローンを一か所に集約する. 本問題では, Java の継承, 移譲の概念を理解することを目的としている．コード 5.4 に問題コードの例を示す.

コード 5.4: クラス作成によるコードクローン除去の問題例

```
 1 public class Main {
 2     public static void main(String[] args) {
 3         Dog dog = new Dog();
 4         dog.say();
 5         Cat cat = new Cat();
 6         cat.say();
 7     }
 8 }
 9 class Dog{
10     String cry = "Bow-wow";
11     public void say() {
12         for(int i = 0; i < 3; i++){
13             System.out.println(cry);
14         }
15     }
16 }
17 class Cat{
18     String cry = "Meowing";
19     public void say() {
20         for(int i = 0; i < 3; i++){
21             System.out.println(cry);
22         }
23     }
24 }
```

本問題コードで除去すべきコードクローンは, 12 行目から 14 行目, 20 行目から 22 行目である. その解答コードの例をコード 5.5 に示す.

コード 5.5: クラス作成によるコードクローン除去の問題解答例

```
 8 ... ( 略 ) ...
 9 class Dog extends Animal{
10     Dog(){
11         cry = "Bow-wow";
12     }
13 }
14 class Cat extends Animal{
15     Cat(){
16         cry = "Meowing";
17     }
18 }
19 class Animal{
20     String cry = "";
```

```
21          public void say() {
22                  for(int i = 0; i < 3; i++){
23                          System.out.println(cry);
24                  }
25          }
26  }
```

コード 5.5 では, Animal クラスを新たに作成し, Dog, Cat クラスがそれを継承することでコードクローンを除去している. また, 新規にクラスを作成しなくても, 例えば Dog クラスを親クラス, Cat クラスを子クラスにすることで, 除去することも可能である. この場合, コードは短くなる. 更に, 継承を使用せず, 親クラスではないクラスにメソッドを移動し, コードクローンを除去できる場合も正解となる.

5.3.6 テンプレートメソッドの形成による除去

本問題では複数の子クラスのメソッドにおいて, 処理の手順は同じであるが詳細な処理内容は異なっている場合に, 処理の手順を共通化することで, コードクローンが除去される. 本研究では, このリファクタリング技法に相当する手法として, Java のデザインパターン [?] を用いることとする. これにより, デザインパターンの学習にも貢献できる. コード 5.6 にその例を挙げる.

コード 5.6: テンプレートメソッドの形成による除去問題

```
46  ... ( 略 ) ...
47  public void trace(Lines g){
48          Lines p = g;
49          while(p!=null){
50                  System.out.println(p.getContent());
51                  if(p.child!=null){
52                          trace(p.getChild());
53                  }
54                  p = p.getNext();
55          }
56  }
57  public void measureLength(Lines g){
58          Lines p = g;
59          while(p!=null){
60                  System.out.println(p.getContent().length());
61                  if(p.child!=null){
62                          trace(p.getChild());
63                  }
64                  p = p.getNext();
65          }
66  }
67  ... ( 略 ) ...
```

コード 5.6 は, 図 5.1 に示す, 木構造をトレースするプログラムである. 図 5.1 では, 双方向チェーンで連結した Lines クラスの各オブジェクトが, 子チェーンとして Lines クラ

スのチェーンを持つことができる木構造が示されている．これにより, 例えば, プログラムのステートメントを表すことができる．その場合, 各節はステートメントを表す．

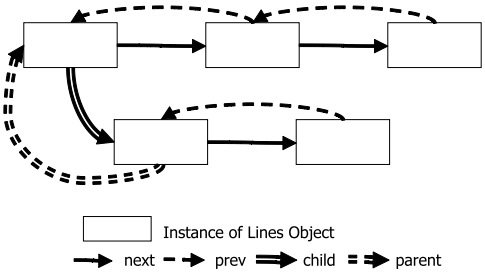

図 5.1: プログラムのステートメントを表す木構造 Lines

コード 5.6 の下線部は, コードクローンとして除去すべきコード断片を示している．ここでの機能は再帰を用いたトレース処理である．具体的には, 以下の処理を行う．

1. 指定されたノードから順に探索内容の処理
2. 子ノードがあれば, その子ノードに対して再帰処理
3. チェーンで繋がれた次のノードを処理

2 つのメソッド trace, measureLength は, ほぼ同じ内容であるが, 50 行目と 60 行目が異なっている．そのため, コード 5.7 に示すように, デザインパターンの 1 つである，*Visitor* パターンを用いる．*Visitor* パターンでは, トレースアルゴリズム（コードクローン）と各ノードに対する処理を別のメソッドに分離して記述することができるため, コードクローンの除去が可能となる．

コード 5.7: テンプレートメソッドの形成による除去問題解答例

```
46  ... ( 略 ) ...
47  abstract class Visitor{
48          abstract void visit(Lines l);
49  }
50  public void trace(Lines g){
51          Visitor visitor = new Visitor(){
52                  void visit(Lines lines){
53                          System.out.println(lines.getContent());
54                  }
55          };
56          g.accept(visitor);
57  }
58  public void measureLength(Lines g){
59          Visitor visitor = new Visitor(){
60                  void visit(Lines lines){
61                          System.out.println(lines.getContent().length());
62                  }
63          };
```

```
64          g.accept(visitor);
65  }
66  public void accept(Visitor v){
67          Lines p = this;
68          while(p!=null){
69                  v.visit(p);
70                  if(p.child!=null){
71                          p.child.accept(v);
72                  }
73                  p = p.getNext();
74          }
75  }
76  ... ( 略 ) ...
```

5.3.7　クローン探査ツール

今回, クローン探査ツールとして, Java コードのスタティック解析ツール PMD のアドオンである, *Copy/Paste Detector (CPD)* を採用した. CPD では, 変数やリテラルの違いの無視や, コードクローンとみなす最小トークン数をパラメータで指定できる. 教員による問題作成の際にも, 作成した問題がその意図に一致するように, CPD を使用しながら, そのパラメータを編集する. 編集されたパラメータは, 作成された問題毎に, データベースに保存されている（実際は, テストコード内に記述）.

5.4　3 段階学習法の提案

本章では, コードクローン除去問題において，学生が出題意図を読み取れず手つかずに終わることの内容に，メソッド作成課題を取り上げ,（1）解法の理解,（2）問題コード中のコードクローンの指摘,（3）解答コードの作成，の 3 段階で学習を進めるコードクローン除去問題の 3 段階学習法を提案する.

5.4.1　2 種類のメソッド作成によるコードクローン除去

メソッド作成によるコードクローン除去の方法を，メソッド抽出とパラメータ化に区別して説明する.

5.4.1.1　メソッド抽出による除去

本手法では, コードクローンを含むコードのメソッドの一部を新たなメソッドとして抽出し, コードクローンをそこにまとめることで除去を行う. コード 5.8 に本手法の学習を行うためのコード例を示す.

コード 5.8: メソッド抽出による除去学習用コード

```
1  public class CctestEx01 {
2      public static void main(String[] args) {
3          dogSay();
4          catSay();
5      }
6      static void dogSay() {
7          System.out.println("Hello.");
8          System.out.print("This is ");
9          System.out.println("Dog.");
10     }
11     static void catSay() {
12         System.out.println("Hello.");
13         System.out.print("This is ");
14         System.out.println("Cat.");
15     }
16 }
```

コード 5.8 の 7-8 行目と 12-13 行目は同一のステートメントであり，コードクローンである．その除去は，それらを含むメソッドを作成し，そのメソッドを呼び出すことで行う．コード 5.9 に解答例を示す．

コード 5.9: メソッド抽出による除去解答例

```
1  class C026 {
2      public static void main(String[] args) {
3          dogSay();
4          catSay();
5      }
6      static void dogSay() {
7          sayHello();
8          System.out.println("Dog.");
9      }
10     static void catSay() {
11         sayHello();
12         System.out.println("Cat.");
13     }
14     static void sayHello() {
15         System.out.println("Hello.");
16         System.out.print("This␣is␣");
17     }
18 }
```

5.4.1.2 パラメータ化による除去

本手法では, メソッド内で利用されている変数やリテラルを引数に変更することで, コードクローンの除去を行う．ここでは, よく似た振る舞いをするものの, 異なる値を利用している複数のメソッドがある場合, その値を引数として受け取るメソッドにまとめることで, コードクローンの除去が可能となる.

コード 5.9 の 8 行目と 12 行目は，出力文の引数の文字列が異なっているのみで，同様の振る舞いをする．そこで，この文字列を受け取るメソッドでそれを引数とすることでコードクローンの除去を行う．コード 5.10 に解答例を示す．

コード 5.10: パラメータ化による除去

```
1  public class CctestEx02 {
2      public static void main(String[] args) {
3          dogSay();
4          catSay();
5      }
6      static void dogSay() {
7          sayHello("Dog.");
8      }
9      static void catSay() {
10         sayHello("Cat.");
11     }
12     static void sayHello(String name) {
13         System.out.println("Hello.");
14         System.out.print("This␣is␣");
15         System.out.print(name);
16     }
17 }
```

コード 5.10 は, テストコードか仕様で dogSay() メソッドと catSay() メソッドの作成が明示されていなければ，更に，コード 5.11 のように改善できる．

コード 5.11: さらに改善されたパラメータ化による除去

```
1  public class CctestEx02 {
2      public static void main(String[] args) {
3          sayHello("Dog.");
4          sayHello("Cat.");
5      }
6      static void sayHello(String name) {
7          System.out.println("Hello.");
8          System.out.print("This␣is␣");
9          System.out.print(name);
10     }
11 }
```

5.4.2 3 段階学習法の提案

上記，メソッド生成課題のコードクローン除去を対象として，3 段階学習法を提案する．

5.4.2.1 解法の理解

まず，学生にメソッド生成課題の解法（コードクローン除去法）の理解を促すために，コードクローンを含むコード（問題コード）とその除去後のコードを並べて提示する．こ

のとき，除去後のコードを理解して貰うために，エレメント空欄補充問題として与える．コード 5.8 のコードに対する，除去後のコードのエレメント空欄補充問題の例をコード 5.12 に示す．学生は，それらの空欄を埋めることで本コードに対する理解を深めることが期待される．

コード 5.12: コードクローン除去問題のためのエレメント空欄補充問題（ステップ 1）

```
 1  class CctestEx01Better {
 2      public static void main(String[] args) {
 3          dogSay();
 4          catSay();
 5      }
 6      static void dogSay() {
 7          sayHello();
 8          System.out.println("Dog.");
 9      }
10      static void catSay() {
11          [ 1 ]();
12          System.out.println("Cat");
13      }
14      static void [ 2 ]() {
15          [ 3 ].out.println("Hello.");
16          System.out.[ 4 ]("This␣is␣");
17      }
18  }
```

5.4.2.2 問題コード中のコードクローンの指摘

次に，以下の画面を用いて，学生に問題コード中のコードクローン部分を行単位で指定させる．学生はコードクローンとなっている行にチェックマークを記入し，解答する．解答は，サーバにおいて，予め正解として準備された行番号と比較して採点を行う．

コード 5.13: コードクローン除去問題のためのコードクローン指摘問題（ステップ 2）

```
 1  □ public class CctestEx02 {
 2  □       public static void main(String[] args) {
 3  □           dogSay();
 4  □           catSay();
 5  □       }
 6  □       static void dogSay() {
 7  □           System.out.println("Hello.");
 8  □           System.out.print("This␣is␣");
 9  □           System.out.println("Dog.");
10  □       }
11  □       static void catSay() {
12  □           System.out.println("Hello.");
13  □           System.out.print("This␣is␣");
14  □           System.out.println("Cat.");
15  □       }
```

```
16  □ }
```

5.4.2.3 解答コードの作成

最後に，学生に従来のコード作成問題と同様の形式で，解答コードの作成を行わせる．但し，コードクローンを含むコード（問題コード）の提示は行う．

5.4.3 解答コードの採点機能

3 段階目の解答コードに対する採点は, 以下の 3 段階で行わる．

1. 動作仕様の検証
2. コードクローン除去の検証
3. コード長の検証

5.4.4 動作仕様の検証

動作仕様の検証は，従来のコード作成問題と同様，JUnit 上でテストコードを用いて行う．コード 5.14 に**プログラム 1** の問題の解答コードの採点のためのテストコードを示す．ここでは，従来のコード作成問題と同様，期待される出力と実際の出力の比較を行い，プログラムの動作仕様が変更されていないかテストする．

コード 5.14: コードクローン除去問題採点用テストコード

```
1  public class C026Test {
2      //@ Test
3      public void testMain() {
4          PrintStream p = System.out;
5          ByteArrayOutputStream bw = new ByteArrayOutputStream();
6          System.setOut(new PrintStream(bw));
7          C026.main(null);
8          System.setOut(p);
9          assertEquals("Hello."+System.lineSeparator()+"This␣is␣Dog."+
       System.lineSeparator()+"Hello."+System.lineSeparator()+"This␣is␣Cat."+
       System.lineSeparator(), bw.toString());
10     }〜略〜
```

5.4.4.1 コードクローン除去の検証

コードクローン除去の検証は，Java コードのスタティック解析ツール PMD のアドオンである, Copy/Paste Detector (CPD) を用いて行う. CPD では, 変数やリテラルの違いの無視や, コードクローンとみなす最小トークン数をパラメータで指定できる. 教員による

コードクローン除去問題の作成の際に, 検出するコードクローンが教員の意図に一致するように，CPD のパラメータを設定する. 設定されたパラメータは, 作成された問題のテストコード内に記述される.

この CPD によるコードクローン検出を，JUnit でのテスト中に行い，検出結果をメッセージとして出力するため，pmd-core-5.5.3,pmd-java-5.5.3,jaxen-1.1.6 を使用して，CPD のラッパークラスを準備した [36].

コード 5.15 に，コードクローン検出のためのテストコードを示す．4 行目で，検出対象のコードクローンの長さをトークンで指定する．5 行目で，このメソッドが動作するクラスファイルを元にソースコードを読み込んでいる．

コード 5.15: コードクローン検出のためのテストコード

```
1  ～略～
2  public void testCPD() {
3          CPDWrapper cw = new CPDWrapper();
4          cw.setMinimumTileSize(20);
5          cw.setTargetClass(this.getClass());
6          cw.cpd();
7          assertThat("Code␣Clones␣are␣found.", cw.matches, cw.hasClone(0));
8          }～略～
```

コード 5.16 にコード 5.15 による実行結果の例を示す．1-2 行目はメッセージで，テストしたソースコードにコードクローンがあることを示している．4 行目はコードクローンのトークン数，5 行目はコードクローンの数．6-7 行目はコードクローンの開始位置，8-10 行目はコードクローンの内容を示している．

コード 5.16: コードクローンを含んだ解答に対するテスト結果

```
1  java.lang.AssertionError: Code Clones are found.
2  Expected: TO DELETE CODE CLONES BELLOW EITHER
3  Match:
4  tokenCount = 22
5  marks = 2
6  \Users～略～\ccds\src\cc26\C026.java(10)
7  \Users～略～\ccds\src\cc26\C026.java(16)
8  System.out.println("Hello.");
9  System.out.print("This␣is␣");
10 System.out.println("Dog.");
11 but: was <java.util.ArrayList$Itr5c0369c4>～略～
```

5.4.4.2 コード長の検証

コード 5.17: コード長の検証のためのテストコード

```
1  ～略～
2  public void testLength() {
3          CPDWrapper cw = new CPDWrapper();
4          cw.setMinimumTileSize(3);
```

```
 5         cw.setTargetClass(this.getClass());
 6         cw.cpd();
 7         assertThat("Make␣code␣shorter.", cw.getTotalLineOfCode(), new
       BaseMatcher(){
 8               int limit = 88;
 9               public boolean matches(Object arg0) {
10                     return (limit>(int)arg0);
11               }
12               public void describeTo(Description description) {
13                     description.appendText("shorter␣code␣less␣than␣"+limit);
14               }
15         });
16   }
```

学生によっては，コードクローン除去を行うために，それらの形を見かけ上，変化させるだけで，更に冗長となるコードを作成する場合が考えられる．それを防止するために，更にコード長の検証を行う．コード 5.17 に，コード長の検証のためのテストコードを示す．8 行目は問題コードの長さであり，学生による解答コードがこれよりも長くなった場合，コード 5.18 の表示を行う．2 行目に問題コードのトークン数，3 行目に解答コードのトークン数が表示される．

コード 5.18: 解答コードが長くなった場合のテスト結果

```
1  java.lang.AssertionError: Make code shorter.
2  Expected: shorter code less than 88
3  but: was <90>
4  at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)〜略〜
```

5.5 評価実験

本章では, コードクローン除去問題について，まず 4 種類の問題に対して 3 段階学習法を用いず，試行をおこなう．次に類似の条件でメソッド作成課題について 3 段階学習法を試行する．

5.5.1 4 種類のコードクローン問題の評価

5.5.1.1 評価方法

本実験の被験者は, 研究室内の学生 7 名で, Java 言語に対する理解度は, 現在 Java プログラミング授業履修中の学生 2 名, 研究で Java プログラミングを行っている学生 5 名である. また, 課題には, 5.3.2 で示した, コードクローン除去問題の 4 種類の例題から 1 つずつを選んだ（表 5.3）.

表 5.1: 問題のテーマと正解数

問題番号	テーマ	正解者数
1	文法適正化によるコードクローン除去	7
2	メソッド作成によるコードクローン除去	4
3	クラス作成によるコードクローン除去	2
4	テンプレートメソッドによる除去	0

5.5.1.2 評価結果

表 5.3 の各問題の正解者数より, 問題が難しくなるにつれ, 正解者数が減少していることがわかる. 特に, 問題 2 ～ 4 は，初学者への出題に適しているとは言い難く，さらに工夫が必要である.

次に, 問題解答後に実施した, これらの学生へのアンケート結果を, 表 5.5.1.2 に示す. ここでは, 各質問に対して, 5 を「とても思う」, 1 を「全く思わない」として, 5 段階で答えてもらった.

表 5.2: アンケート結果

質問	5(yes)	4	3	2	1(no)
冗長なコードは問題と思うか	4	3	0	0	0
クラスの理解が増したか	4	2	1	0	0
デザインパターンの理解が増したか	1	4	2	0	0

5.5.1.3 考察

表 5.5.1.2 より, 学生全員が冗長なコードに対する問題意識はあるが, クラスの概念を十分理解していないために, クラスを利用した問題解決には至っていないことがわかる. 今後, コードクローン除去問題を解くことで, クラスの理解が進むことが期待される. 更には, クラスの理解が不十分な段階では, デザインパターンの学習が困難であると言える. 今後, コードクローン除去問題においては, 1 つの種類の問題における学生の理解度を確認した上で, 次のより難度の高い種類の問題を提示することが必要といえる.

5.5.2 3 段階学習法の評価

次に，メソッド作成課題を対象とした 3 段階学習法の評価を行う.

5.5.2.1 評価方法

本実験の被験者は, 研究室内の学生 7 名である．Java に対する理解度としては, 初学者として現在 Java プログラミング授業履修中の学生 2 名, 研究で Java プログラミングに習熟している学生 5 名である.

評価実験に使用した問題は，メソッド作成課題における,「メソッド抽出による除去」,「パラメータ化による除去」のそれぞれにおいて,「解法の理解」,「コードクローンの指摘」,「解答コードの作成」の 3 段階を用意した．但し,「解法の理解」では，それ以降の段階での正解コードを示すことを防ぐために，それらとは異なるコードを使用した.

5.5.2.2 メソッド抽出による除去の問題

「解法の理解」では，コード 5.12 を提示し,「コードクローンの指摘」,「解答コードの作成」では，コード 5.19 に示す処理の前後に時刻を表示するコードを提示した.

コード 5.19: ログ時刻表示メソッド抽出による除去の問題

```
1  import java.io.File;
2  import java.util.Date;
3  public class LapTime {
4          static Date dt;
5          public static void main(String[] args) {
6                  System.out.print("Start␣at␣:␣");
7                  dt = new Date();
8                  System.out.println(dt.toString());
9                  int wk = 1;
10                 for(int i = 10; i > 0; i−−) {
11                         File file = new File("C:\\");
12                         double total = (double)file.getTotalSpace();
13                 }
14                 System.out.print("End␣at␣:␣");
15                 dt = new Date();
16                 System.out.println(dt.toString());
17         }
18 }
```

5.5.2.3 パラメータ化による除去の問題

「解法の理解」では，コード 5.11，それ以降では，コード 5.20 に示す，反復回数のパラメータ化を必要とするコードを提示した.

コード 5.20: 反復回数のパラメータ化による除去の問題

```
1  public class CctestEx02 {
2          public static void main(String[] args) {
3                  threeTimesSay();
4                  twoTimesSay();
5          }
```

```
 6         static void threeTimesSay() {
 7             for(int i = 0; i < 3; i++) {
 8                 System.out.println("Baw␣");
 9             }
10         }
11         static void twoTimesSay() {
12             for(int i = 0; i < 2; i++) {
13                 System.out.println("Baw␣");
14             }
15         }
16 }
```

5.5.3 評価結果

表 5.3 に本評価実験の結果を示す．今回対象とした，メソッド生成課題の 2 種類の問題の 3 段階それぞれにおける正解者数を示す．表**??**の結果に比べ，解答コード作成の正解率が向上していることから，提案する 3 段階学習法により，コードクローン除去問題のメソッド作成課題に対する解法の理解が高まったと言える．

表 5.3: 問題のテーマと正解数

	例題	コードクローン個所指摘問題	コード作成
メソッド抽出	7	7	6
パラメータ化	7	2	6

一方，今回出題した「パラメータ化による除去」の問題に対して，コードクローン個所を正確に指摘できた者は少なかった．この問題では，コード 5.20 の 7-9 行目と 12-14 行目を指摘できれば正解となる．しかし，多くの解答では，コード 5.21 に示すように，8 行目と 13 行目のみを指摘していた．

コード 5.21: 誤りの多かったコードクローン箇所指摘問題解答

```
7  □ for(int i = 0; i < 3; i++){
8  √       System.out.println("Baw");
9  □ }
```

コードクローンを探す際に学生が意識しているステートメントは,「分岐文」,「反復文」といった制御文ではなく，処理を記述したステートメントのみであるが，多くの学生は，それでもコードクローン除去問題の正解コードを解答している．その原因の分析と対策が今後の課題である．

5.6 まとめ

本研究では, JPLAS の新たな問題として提案している**コードクローン除去問題**において，メソッド生成課題の 2 種類の問題を対象として，解法の理解，問題コード中のコードクローンの指摘，解答コードの作成，の 3 段階学習法を提案した．評価として研究室の学生 7 名に本提案手法を適用した結果，従来よりも正解率が改善されたが，同時に，必ずしも正しくコードクローンを指摘できていないことが分かった．

今後は，学生のコードクローンの見逃しに対する対処方法を検討するとともに，今回対象としなかった，クラス生成およびテンプレートメソッド使用によるコードクローン除去の課題への 3 段階学習法の適用を検討する．

第6章　関連研究

本章では，本研究を進めるにあたり，参考とした関連研究について紹介する．

6.1　ソフトウェアアーキテクチャの関連研究

JPLAS のソフトウェアアーキテクチャに関連した研究を紹介する．

- **Struts framework and model-view-controller design pattern**[16]
  文献 [16] では，IBM ナレッジ・センターで開発された，Struts フレームワークとモデル・ビュー・コントローラの設計パターンが紹介されている．ここでは，HTML ファイルを使用したコンテンツの生成方法や，ビジネスロジックとコンテンツプレゼンテーションを分離できる，モデル 1 とモデル 2 と呼ばれる 2 つの JSP モデルが示されている．モデル 1 は，JSP ページと Java Bean コードのみを使用するのに対し，モデル 2 はアプリケーションソフトウェアアーキテクチャを開発するのに役立つ MVC モデルとなっている．Struts フレームワークは，モデル 2 に沿っている．

- **A modular method and framework for Web application development using XSLT**[37]
  文献 [37] では，XSLT を使用して Web アプリケーションのフロントエンドを実装する方法が提案されている．ここでは，フレームワークとして実装されている．すなわち，サーブレットクラスのコレクションがターゲット Web ソフトウェアのコンポーネントとなり，全体のロジックは XSLT で記述される．

- **Realtime distributed MVC architecture using Ajax,**[38]
  文献 [38] では，リアルタイム Web アプリケーションを実現するための *Web Distributed MVC*（*WD-MVC*）アーキテクチャを提供しました。 *WD-MVC* は Ajax を使用して Web サーバーからブラウザにメッセージを非同期で伝えます。

- **Web application framework for end-user-initiative development**[39]
  文献 [39] では，エンドユーザ主導の開発プロジェクトのために em EcoFW が提案されている． em EcoFW では，ビューとモデルを分離するだけでなく，Ajax と JSON フォーマットを使って，コンポーネントごとにビューを実現する em Struts のようなフレームワークのコントローラ設定に重点を置いている．

6.2 ステートメント空欄補充問題の関連研究

ステートメント空欄補充問題に関連した研究を紹介する．

- **Tukra: An Abstract Program Slicing Tool**[40]
  抽象プログラムスライシングアルゴリズムの実用的な評価を可能にするツールとして，*Tukra* が紹介されている．ステートメント間の関連性，セマンティックデータ間の依存，条件依存性の 3 つの概念の組み合わせを利用している．また，**プログラムスライス**について簡便に記述している．

- **Java バーチャルマシンを利用した動的依存関係解析手法の提案** [41]
  プログラムデバッグを効率的に行う手法の一つとしてのスライスにおいて， DC スライスを Java へ適用する手法を提案している．ここでは，Java の実行時決定要素を JVM を用いて考慮し，バイトコードに対しての DC スライスを計算するために，コンパイラの機能拡張を行っている．

- **変数依存グラフを用いたスライス計算アルゴリズムとその計算量** [42]
  変数に着目したスライスの計算アルゴリズムと，その計算量の一覧表が示されている．

6.3 コードクローン除去問題の関連研究

コードクローン除去問題に関連した研究を紹介する．

- **コードクローンに基づくレガシーソフトウェアの品質の分析** [43]
  文献 [43] では，コードクローンの影響度が，信頼性・保守性との関係で定量的に明らかされている．分析の結果，コードクローンを含むモジュールの信頼性はコードクローンがある程度の大きさを超えた場合に，非常に低くなる．また，モジュールに含まれるコードクローンのサイズが大きいほど，その更新数がより大きくなる傾向にあるとしている．

- **コードクローンを対象としたリファクタリング** [44]
  文献 [44] では，コードクローンとは, ソースコード中に存在する互いに一致, もしくは類似したコード片であると定義されている．コードクローンの存在は, ソフトウェアの開発および保守に悪影響を与える恐れがあるとしている. ここでは，コードクローンを取り除くためのリファクタリング方法と，近年の研究成果について紹介している.

- **オープンソース開発におけるコードクローン含有率の収束傾向に関する調査** [45]
  文献 [45] では，コードクローンの適切な分量について検討するため，ソフトウェアの開発・保守の進行に伴うクローン含有率の変化を調査している．その結果，20%程度に収束する傾向にあることを報告している．すなわち，コードクローンは保守を

困難にする要因の一つであるが，全てのクローンが有害とは限らず，全てを除去することが適切とは言えない．

- **Clone Detection Using Abstract Syntax Trees**[46]
  文献 [46] では，抽象構文木を用いて，プログラムソースコード中の任意のプログラム断片に対する，完全クローン，および，ニアミスクローンを検出する簡単で実用的な方法が提示されている．

第7章　結論

本研究では，Javaプログラミング学習支援システムJPLAS(Java Programming Learning Assistant System)において，JPLASのソフトウェアアーキテクチャの提案と，２つの新しい問題形式の提案を行った．

ソフトウェアアーキテクチャの提案では，従来のJPLASの実装における問題点を示し，その解決を目的とした，MVCモデルに沿った実装を可能とする，コード記述ルールを提案した．本ルールでは，データベース処理のための抽象クラスの採用，レスポンシビリティ・チェーンデザインパターンを用いた様々な正誤判定機能選択の実現，JSP + Ajaxによる画面遷移の実現，などを規定した．本提案に沿って，JPLASの学生支援機能を再構築し，その前後のJPLAS実装のファイル数の比較と，２種類の新機能拡張における追加ファイル数の評価を行った．

ステートメント補充問題の提案では，まず，PDGを用いた，空欄とするコアステートメントの抽出アルゴリズムを提案した．そのために，Java学習で学ぶべき2つの要素を定義し，それぞれでのコアステートメント抽出手順を明らかにした．提案アルゴリズムで生成した問題を用いた評価により，メソッド内要素に対するステートメント補充問題は，コードリーディング力の育成に有効であると言えた．また，Javaプログラミングの学習者によるコアステートメントの抽出結果と比較し，80%以上の一致を見た．

コードクローン除去問題の提案では，**コードクローン**除去の４種類の手法を明らかにした上で，それぞれに対する問題の定義を行った．また，コードクローン除去手法の理解の支援のために，3段階学習法を提案した．評価として研究室の学生７名に本提案手法を適用した結果，従来よりも正解率が改善されたが，同時に，必ずしも正しくコードクローンを指摘できていないことが分かった．

最後に，本研究の今後の課題について述べる．まず，JPLASの実装において，画像ファイルを含む問題文やJavaバイトコードによる解答提出といった新しい機能への対応が挙げられる．また，コードクローン除去問題において，現在，1種類のコードクローン除去手法にのみ，3段階学習法の実装・評価を終えており，残る2種類の手法への実装・評価が挙げられる．

謝辞

本研究の全般に渡り，深いご理解を頂き，本論文の主査を務めて頂きました，岡山大学大学院自然科学研究科産業創成工学専攻　舩曳信生教授には，心より感謝致します．本論文をまとめるにあたり，ご多忙の中，熱心かつ親切にご指導賜りましたことに対しまして，厚く御礼申し上げます．

本論文の副査を務めて頂きました，岡山大学大学院自然科学研究科産業創成工学専攻　田野哲教授，野上保之教授には，本論文を仕上げるために様々なご指導を頂きましたことを，心より感謝致します．

また，本研究を進めるにあたって，数々の有益なご指導を頂きました，岡山大学大学院自然科学研究科産業創成工学専攻　栗林稔准教授，國立臺灣師範大學　電機工程學系 高文忠教授，中国学園大学/中国短期大学の福森護教授，橋本和久教授には，心より感謝致します．

種々の御協力と御助言を頂きました，分散システム構成学研究室の皆様には，心から感謝申し上げます．特に，同研究室の Khin Khin Zaw 氏には，プログラム作成や評価実験におきまして，様々なご協力を頂戴しましたことに感謝致します．また，事務手続きや研究室生活の面でお世話を頂きました，河端敬子事務補佐員に感謝致します．

博士課程在学中，友人の大橋美佐子氏の存在が，研究を進めていく上で大きな励ましとなりました．氏へエールを送るとともに，深く感謝申し上げます．

最後に，本研究の遂行に際し，常に著者を叱咤激励下さり，温かく見守って下さった，家族に，深く感謝申し上げます．

参考文献

[1] 吉田英輔, 角川裕次, "テスト駆動開発に基づくプログラミング学習支援システム：初心者開発者のためのセルフトレーニングアーキテクチャ," 信学技報, SS2005-44, pp. 27-32, 2005.

[2] N. Funabiki, Y. Matsushima, T. Nakanishi, K. Watanabe, and N. Amano, "A Java programming learning assistant system using test-driven development method," IAENG Int. J. Computer Science, Vol. 40, No.1, pp. 38-46, Feb. 2013.

[3] N. Funabiki, Y. Korenaga, Y. Matsushima, T. Nakanishi, and K. Watanabe, "An online fill-in-the-blank problem function for learning reserved words in Java programming education," Proc. FINA 2012, pp. 375-380, Mar. 2012.

[4] N. Funabiki, Y. Korenaga, T. Nakanishi, and K. Watanabe, "An extension of fill-in-the-blank problem function in Java programming learning assistant system," Proc. R10-HTC2013, pp. 95-100, Aug. 2013.

[5] Tana, N. Funabiki, and N. Ishihara, "A proposal of graph-based blank element selection algorithm for Java programming learning with fill-in-blank problems," Proc. IMECS 2015, pp. 448-453, Mar. 2015.

[6] N. Funabiki, T. Ogawa, N. Ishihara, M. Kuribayashi, and W.-C. Kao, "A proposal of coding rule learning function in Java programming learning assistant system," Proc. VENOA-2016, pp. 561-566, July 2016.

[7] N. Funabiki, Tana, K. K. Zaw, N. Ishihara, and W.-C. Kao, "Analysis of fill-in-blank problem solutions and extensions of blank element selection algorithm for Java programming learning assistant system," Proc. WCECS 2016, pp. 237-242, Oct. 2016.

[8] Tana, N. Funabiki, T. Nakanishi, and N. Amano, "An improvement of graph-based fill-in-blank problem generation algorithm in Java programming learning assistant system," Proc. Int. Work. ICT, pp. 1-4, Dec. 2013.

[9] K. K. Zaw, N. Funabiki, and W.-C. Kao, "A proposal of value trace problem for algorithm code reading in Java programming learning assistant system," Inform. Eng. Express, Vol. 1, No. 3, pp. 9-18, Sep. 2015.

[10] 石原信也, 舩曳信生, 中西透, “Java プログラミング学習支援システムにおけるステートメント補充問題機能の実装,” 信学技報, ET2013-98, pp. 35-40, Mar. 2014.

[11] 石原信也, 舩曳信生, 栗林稔, "Java プログラミング学習支援システムにおけるコードクローン除去問題の提案," 信学技報, SS2016-65, pp. 47-52, Jan. 2017.

[12] J. J. Garrett, "Ajax: a new approach to Web applications", http://zqsmm.qiniucdn.com/data/20050225112101/index.html, Feb. 18, 2005.

[13] JUnit, http://www.junit.org/

[14] 渡辺修司, “JUnit 実践入門体系的に学ぶユニットテストの技法,” 技術評論社, p.308, 2012.

[15] K. Beck, *Test-driven development: by example*, Addison-Wesley, 2002.

[16] Struts framework and model-view-controller design pattern, http://www.ibm.com/support/knowledgecenter/SSRTLW_6.0.1/com.ibm.etools.struts.doc/topics/cstrdoc001.html.

[17] https://struts.apache.org/

[18] jQuery, http://jquery.com/.

[19] Bootstrap, https://getbootstrap.com/.

[20] SkyBlue, https://stanko.github.io/skyblue/.

[21] N. Ishihara, N. Funabiki, M. Kuribayashi, and W.-C. Kao, "A proposal of software architecture for Java programming learning assistant system," Proc. AINA-2017, pp. 64-70, Mar. 2017.

[22] The BLOB and TEXT Types, https://dev.mysql.com/doc/refman/5.7/en/blob.html.

[23] Commons FileUpload, https://commons.apache.org/proper/commons-fileupload/.

[24] 小川卓也, 舩曳信生, 栗林稔, 石原信也, 天野憲樹, "Java プログラミングにおけるリーダブルコード学習ツールの提案," 信学技報, SS2015-50, pp. 35-40, Jan. 2016.

[25] N. Funabiki, H. Masaoka, N. Ishihara, I-W. Lai, and W.-C. Kao, "Offline answering function for fill-in-blank problems in Java programming learning assistant system," Proc. ICCE-TW 2016, pp. 324-325, May 2016.

[26] A Survey of Program Slicing Techniques, http://www.franktip.org/pubs/jpl1995.pdf.

[27] 柏原昭博, 寺井淳裕, 豊田順一, "いかにプログラム空欄補充問題を作るか?," 信学技報, Vol. . 99, No. 81, pp.9-16, May 1999.

[28] N. Ishihara, N. Funabiki, and W.-C. Kao, "A proposal of statement fill-in-blank problem using program dependence graph in Java programming learning assistant system," Inform. Eng. Express, Vol. . 1, No. 3, pp. 19-28, Sep. 2015.

[29] 中村拓哉, 舩曳信生, 中西透, 天野憲樹, “Java プログラミング学習支援システムのコード作成問題における Javadoc を用いたヒント機能,” 信学技報, ET2013-87, pp. 115-120, Jan. 2014.

[30] 結城浩, “増補改訂版 Java 言語で学ぶデザインパターン入門,” SB クリエイティブ株式会社, Feb. 2014.

[31] フレームの作成 (JFrame クラス), http://www.javadrive.jp/tutorial/jframe/.

[32] Java で 2 分探索のサンプル,http://blog.codebook-10000.com/entry/20140103/1388725200.

[33] Y. Higo, and N. Yoshida, "An introduction to code clone refactoring," JSSST, Computer Software, Vol. 28, No. 4, pp. 43-56, Dec. 2011.

[34] Jeff Atwood, "Why Can't Programmers.. Program?," https://blog.codinghorror.com/why-cant-programmers-program/, Feb. 16, 2007.

[35] "checkstyle," http://checkstyle.sourceforge.net

[36] "PMD - Finding duplicated code," https://pmd.github.io/pmd-5.4.1/usage/cpd-usage.html.

[37] Y. Tani, N. Mitsuda, and T. Ajisaka, "A modular method and framework for Web application development using XSLT," IPSJ Tech. Report, 2003-SE-144, pp. 131-138, Mar. 2004.

[38] T. Nagao, Y. Tsuchiya, S. Morimoto, and Y. Chubachi, "Realtime distributed MVC architecture using Ajax," Trans. IPSJ-PRO, Vol. 48, pp. 200-200, June 2007.

[39] S. Ree and C. Takeshi, "Web application framework for end-user-initiative development," Proc. FIT2011, Vol. 1, pp. 271-274, Sep. 2011.

[40] R. Halder and A. Cortesi, “Tukra: An Abstract Program Slicing Tool,” http://www.dsi.unive.it/~cortesi/paperi/ICSOFT_2012.pdf.

[41] 誉田謙二, 梅森文彰, 大畑文明, 井上, 克郎, "Java バーチャルマシンを利用した動的依存関係解析手法の提案,” 信学技報. SS2002-101, pp. 47-54, Jan. 2002.

[42] 太田剛, 渡辺尚, 水野忠則, "変数依存グラフを用いたスライス計算アルゴリズムとその計算量," 情報処理学会全国大会講演論文集, pp. 69-70, Mar. 1996.

[43] 門田暁人, "コードクローンに基づくレガシーソフトウェアの品質の分析," 情処論, Vol. 44, No. 8, Aug. 2003.

[44] 肥後芳樹, 吉田則裕, "コードクローンを対象としたリファクタリング," コンピュータソフトウェア, Vol. 28, No. 4, pp. 43-56, Oct. 2011.

[45] 本田暁, 阿萬裕久, 佐々木隆志, 川原稔, "オープンソース開発におけるコードクローン含有率の収束傾向に関する調査," 信学論 D, Vol. J97-D, No.7, pp.1213-1215, July 2014.

[46] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," Proc. ICSM '98, pp. 368-377, Nov. 1998.

博士論文

Java プログラミング学習支援システムのソフトウェアアーキテクチャと 2 種類の新問題形式の提案

発行日：2018 年 3 月

著者名：石原 信也