模範解答を利用したGUIプログラミング課題の自動採点システム

立石 良生† 井上 潮‡

†東京電機大学 工学研究科 情報通信工学専攻 〒120-8551 東京都足立区千住旭町5番
‡東京電機大学 工学部 情報通信工学科 〒120-8551 東京都足立区千住旭町5番
E-mail: †18kmc10@ms.dendai.ac.jp, ‡inoueu@mail.dendai.ac.jp

あらまし プログラミングの授業が多くの大学で行われている．オブジェクト指向を習得するためにはGUIのプログラミング演習が有効であるが，GUIの演習課題の採点にはマウス，キーボードでの操作が必要なため，非常に手間がかかる．既存のGUI自動テスト手法は特別な環境を必要とし，採点用プログラムの作成にスキルと時間を要する．本研究では，教員が作成した模範解答から採点用プログラムを自動的に生成し，学生が使用する標準的な開発環境で演習課題の採点作業を自動化するシステムを実現した．実際の授業で用いられた演習課題で評価した結果，多数の学生のプログラムを短時間かつ高精度で自動採点できることを確認した．

キーワード GUI，オブジェクト指向，プログラミング課題，自動テスト，JavaFX

1. はじめに

近年，人工知能（AI）やIoTなどのテクノロジーが急速に進化していることからプログラミング教育が重要視されている．文部科学省は，2020年度より小学校でプログラミング教育を必修化することを明示しており，その後中学校と高等学校でも必修化が実施されることになっている[1]．このことからプログラミング言語に興味を示す学生およびプログラミング授業を行う教育機関は増加することが考えられる．

初心者向けのプログラミング言語としてJavaが用いられることが多い．Javaの学習ではコマンドラインインタフェース（CLI）のプログラムを作成する他に，オブジェクト指向プログラミングを習得するためにグラフィカルユーザインタフェース（GUI）のプログラムを作成する．プログラミング言語の習得は実際にソースコードを記述することが非常に重要なため，プログラミングの授業において毎回数個の簡単な演習課題が出題される．しかし，数百人分の学生のプログラミング演習課題を手作業で採点するのは非常に手間がかかり，教員の負担が大きい．特に，GUIのプログラミング課題を採点する場合は描画されたレイアウトの確認やマウスでの操作が必要なため，より多くの時間を必要とする上に採点ミスも発生しやすくなる問題がある．そこで本研究では教員が事前に作成した演習課題の模範解答を利用し，学生のGUIプログラミング課題を自動で採点するシステムを開発した．

本論文は以下のような構成になっている．第2章で関連研究を紹介し，従来の自動採点手法の利点と問題点を述べる．第3章では前提条件としてGUIライブラリや自動採点対象のGUI，本システムで自動化する作業について説明する．第4章では我々が提案する自動採点手法を述べる．第5章では実際のプログラミング課題を用いて自動採点システムの動作を示す．第6章では既存手法や手動採点と比較して提案システムの評価を行った結果を示し，第7章で全体のまとめと今後の課題について述べる．

2. 関連研究

Nguyenら[2]は，「GUITAR」と呼ばれるGUIアプリケーション自動テスト用ツールを開発した．これはモデルベースのフレームワークとなっており，GUIテスト対象のコンポーネントやイベントを容易に拡張することができる．これにより，従来の手法と比較して様々なGUIアプリケーションの自動テストに柔軟に対応できるようになった．しかし，手動での作業や誤検出，予期せぬエラーなどが発生する問題が挙げられている．

井上[3]は，学生が作成した演習課題のGUIプログラム を自動でテストするシステムを開発した．これはスクリプトベースで開発されており，JavaFXで記述されたGUIアプリケーションをマウスやキーボードでの操作をせずに自動で採点することができる．また，採点用プログラムはJavaFXのみで記述されているため，特別なスキルを習得せずに作成することができる．しかし，1つの採点用プログラムを作成するのに1時間程度かかることから，多種類のGUIをテストするのには多くの時間を必要とすることが課題として挙げられている．

GUIアプリケーションの自動採点手法は他に手動での操作を記録し，テスト時に自動で再生させる「Capture/Replay方式」[4]，モデルベースを発展させた「パターンベース手法」[5]，[6]など様々な手法が提案されている[7]．しかし，これらの手法では特別な開発環境やスキル，時間を必要とすることが問題となっている[8]．本研究では，自動採点を行うための採点用プログラム作成を簡略化し，手作業での採点と同じ精度で採点ができる手法を開発した．

3. 前提条件

3.1. JavaFX

Java の GUI ライブラリとして代表的なものは Swing と JavaFX が挙げられる．Swing は AWT の拡張ライブラリとしてよく用いられており，Swing を対象とした自動テストも多く提案されていたが[9]，Java8 からは JavaFX が標準 GUI ライブラリとして導入されていることから，JavaFX を対象とした自動テスト手法も提案されている[10]．本研究では JavaFX で作成された GUI プログラムを対象とする．JavaFX での GUI 作成は以下の手順で行う．

① ボタン，ラベルなどのコンポーネント(Control)を作成する．
② コンポーネントにイベントを登録する．
③ コンポーネントをレイアウトペイン(Pane) に配置する．
④ レイアウトペインをシーンにセットする．
⑤ ステージ(ウィンドウ) にシーンをセットする．

JavaFX の GUI 構成の例を図 1 に示す．図 1 のように，レイアウトの情報は木構造になっている．JavaFX ではコンポーネントが多種類用意されており，様々なレイアウトの GUI を作成することができる．開発したシステムで対応しているレイアウトペインとコンポーネントの一覧を表 1，表 2 に示す．

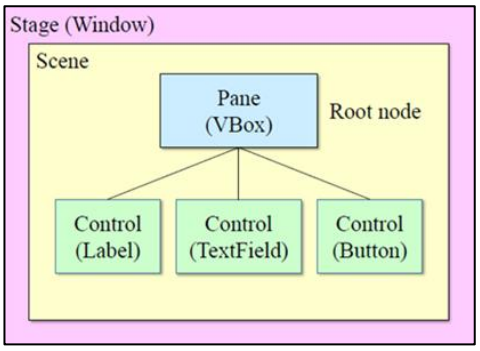

図 1　JavaFX の GUI 構成の例

表 1　対応したレイアウトペイン

名称	説明
HBox	コンポーネントを水平に配置
VBox	コンポーネントを垂直に配置
BorderPane	上，左，右，下，中央の各位置に配置
GridPane	行と列の柔軟なグリッド内に配置
FlowPane	境界で折り返されるフローに配置

表 2　対応したコンポーネント

名称	説明
Label	文字を出力するラベル
TextField	文字を入力できる領域
Button	イベントを登録できるボタン
RadioButton	選択肢の中から 1 つ選ぶボタン
ComboBox	選択肢を提供するリスト
CheckBox	選択肢を提供するボックス

3.2. 自動採点対象の GUI

本研究で対象とする GUI アプリケーションは，プログラミング初心者向けの簡単な GUI アプリケーションとする．具体的には，数字や文字列を入力，選択しボタンを押すと結果が出力されるようなアプリケーションを自動採点の対象とする．図形が出力されるようなアプリケーションは対象外とする．また，本システムは模範解答を利用して自動採点を行うので学生の解答が模範解答と同じレイアウト構造になっている必要がある．演習課題を出題する際に使用するレイアウトペインとコンポーネントを指定する必要がある．

3.3. 自動化する作業

GUI のテストは「静的テスト」と「動的テスト」に分けられる[11]．静的テストでは描画された画面のレイアウトが正しいか確認し，動的テストではイベントの動作が正しく行われるか確認するテストである．本システムでは静的テストと動的テストを自動化する．また，予期せぬエラーを防ぐため最初に静的テストを行い，正しいことが確認できた場合のみ動的テストを行うこととする．

4. 提案手法

4.1. システムの概要

従来のスクリプトベースの自動採点では採点する GUI ごとに採点用スクリプトを作成しなければならなかった．そこで筆者は事前に教員が作成した模範解答のプログラムからレイアウトの情報やイベントを取得し，XML 形式でファイルに保存する手法を提案した．学生の演習課題を自動採点する際は，同様の手法で学生のプログラムからレイアウトの情報を取得し，模範解答の XML と比較することで採点を行う．これにより，演習課題ごとに採点用プログラムを作成する必要がなくなり，模範解答を用意するだけで様々な演習課題の自動採点を行うことができる．また，GUI のレイアウトは階層構造となっているため，XML の形式で格

納するのに適している．

提案システムの構成を図 2 に示す．最初に教員は演習課題の模範解答プログラムを用意する．模範解答からレイアウトの情報やイベントを取得し，XML に自動で書き込む．模範解答の XML ファイルと学生のプログラムを用意すれば静的テスト，動的テストを自動で行い，採点することができる．

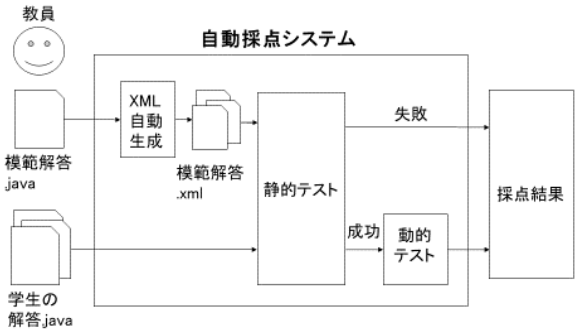

図 2　提案手法のシステム構成

4.2. XML 自動生成手法

ここでは模範解答のプログラムから静的テスト用 XML (StaticTest.xml) と動的テスト用 XML (DynamicTest.xml) を生成する手法について説明する．

静的テスト用 XML の生成は，最初に描画された模範解答の GUI からレイアウト情報をプログラム上で取得する．具体的には，レイアウトペイン，コンポーネントの種類と位置情報，テキスト文を取得し，木構造で格納する．生成された XML に対して 5.2 節で述べる方法で採点対象項目を追記することにより，静的テスト用 XML が生成される．動的テスト用 XML にはレイアウト情報に加えて入出力の値（テストケース）を格納する必要がある．教員はテストケースを用意し，描画された GUI に値を入力してイベントを発生させると，プログラム上で入出力の値を自動取得し，動的テスト用 XML に書き込まれる．この操作を繰り返していくと，様々なテストケースに対応できる．動的テスト用 XML はテストケースの数だけファイルが生成される．XML の要素（タグ）にはコンポーネント，レイアウトペインの名称が格納される．XML の属性には点数（1 点または 0 点），ID，アライメント（レイアウトペインのみ），そして入力，イベント，出力の情報（コンポーネント）が格納される．コンポーネントのテキストは XML のテキストとして格納される．

4.3. 静的テスト手法

学生が作成したプログラミング課題を採点する場合は，静的テスト用 XML を生成するのと同じ手法で学生の GUI からレイアウトの情報を取得する[12]．学生の GUI から取得したレイアウトの情報を静的テスト用 XML と比較することによって採点を行うことができる．静的テストのシステム構成を図 3 に示す．

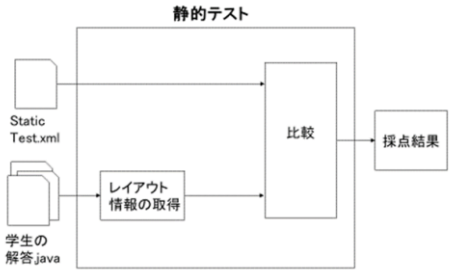

図 3　静的テストのシステム構成

4.4. 動的テスト手法

4.2 節の手法で作成した動的テスト用 XML と学生が作成した GUI アプリケーションを用いて動的テストを行う．動的テスト用 XML から入力値を取得し，学生の GUI に自動で入力する．次に，プログラム上で学生の GUI のイベントを発生させる．GUI に出力された値と動的テスト用 XML に格納されている値を比較することで，学生の GUI アプリケーションが正しく動作しているか確認することができる．動的テストのシステム構成を図 4 に示す．

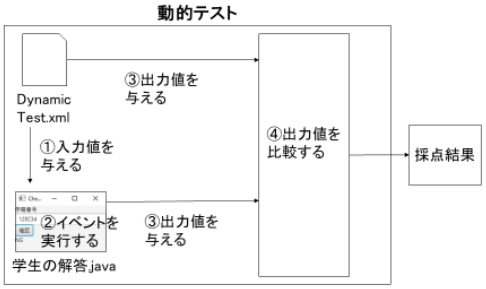

図 4　動的テストのシステム構成

5. システムの実装

5.1. システムの流れ

本システムは自動採点に必要な準備をするための XML 自動生成プログラム，主に学生が使用する自動テストプログラム，複数人の演習課題をまとめて採点する自動採点プログラムの 3 つから構成されている．教員は静的テスト用 XML と動的テスト用 XML を生成し，学生はこれらの XML と自動テストプログラムを用いて自ら作成した GUI の動作確認を行い，教員は自動採点プログラムを用いて学生の GUI を採点するという流れになっている．

5.2. XML 自動生成システムの動作

XML を自動生成するシステムは GUI アプリケーションとなっており，ユーザーが操作しながら XML を生成する．ここで用いる演習課題は図 5 で示す学籍番号確認アプリケーションとする．ユーザーが入力した文字列が「xxECxxx(x は任意の数字)」 となっているか確認するアプリケーションである．

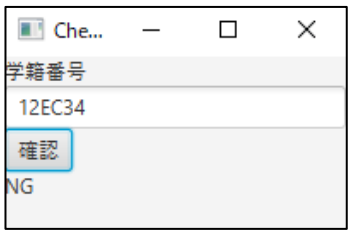

図 5　学籍番号確認アプリケーション

5.2.1. 採点項目の選択

自動生成プログラムを実行すると最初に図 6 のような画面が表示される．これは模範解答の GUI から取得したレイアウト情報をチェックボックスで表示したものである．ユーザーはここで採点したい項目を選択する．下の決定ボタンを押すと図 7 のような静的テスト用 XML ファイルが生成される．チェックボックスで選択した項目には XML の属性として「Score=”1”」と記述される．

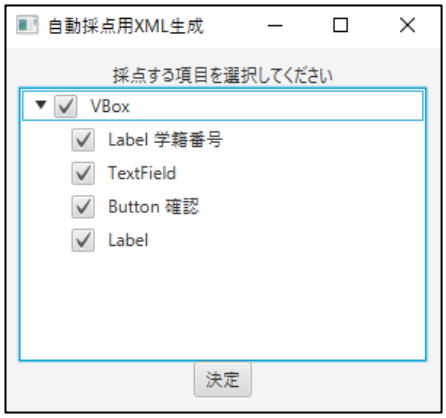

図 6　採点項目選択画面

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Scene ID="0" ScoreMax="5">
    <VBox Alignment="TOP_LEFT" ID="1" Score="1">
        <Label ID="2" Score="1">学籍番号</Label>
        <TextField ID="3" Score="1"/>
        <Button Event="true" ID="4" Score="1">確認</Button>
        <Label ID="5" Score="1"/>
    </VBox>
</Scene>
```

図 7　静的テスト用 XML

5.2.2. 入力，イベント，出力の選択

次に，図 6 と同じような画面で入力，イベント，出力のコンポーネントを選択する画面が順に表示される．ここでは学籍番号を入力するテキストフィールドが入力，確認ボタンがイベント，OK または NG が出力されるラベルが出力となる．イベントを選択すると以下の手順でイベントが発生するように設定される．

① 入力された値を取得する．
② 既に登録されてあるイベントを実行する．
③ 出力された値を取得する

5.2.3. テストケースの作成

入力，イベント、出力を選択して決定ボタンを押すと図 8 のような画面が表示される．ここでは，実際に GUI を動かしながらテストケースを生成する．模範解答の GUI に学籍番号を入力して確認ボタンを押すと，テキストフィールドの値（入力値）を取得し，イベントを実行し，出力されたラベルの文字列（出力値）を取得する．その後，図 9 のような動的テスト用 XML ファイルが生成され，図 8 の画面の右側に入力値と出力値が表示される．選択した入力，イベント，出力のコンポーネントにはそれぞれ「Input=”true”」，「Event=”true”」，「Output=”true”」が XML の属性として与えられる．そして入力と出力の値がテストケースとして XML に記述されている．この操作を繰り返すことで，様々なテストケースを格納した動的テスト用 XML ファイルが作成できる．

図 8　テストケース作成画面

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Scene ID="0" ScoreMax="7">
    <VBox Alignment="TOP_LEFT" ID="1">
        <Label ID="2">学籍番号</Label>
        <TextField ID="3" Input="true">12EC34</TextField>
        <Button ID="4" Event="true">確認</Button>
        <Label ID="5" Output="true">NG</Label>
    </VBox>
</Scene>
```

図 9　動的テスト用 XML

5.3. 自動テストの動作

5.2 節で生成された 2 種類の XML を用いて学生の GUI の自動テストを行う．学生の GUI と XML ファイル，自動テストプログラムを同じディレクトリに格納して自動テストプログラムを記述すると静的テストと動的テストが自動で行われる．自動テストの結果画面を図 10 に示す．静的テストでは，事前に決めた採点項目が模範解答と一致しているか調べ，一致している場合には「OK」と表示され，異なっている場合には「NG」と表示される．静的テストが成功すると，次に動的テストを行う．動的テストでは入力したテストケースの値と出力された値が表示される．この値がテストケースの出力値と一致していれば成功となる．全てのテストケースで正しく動作したことが確認できると，最後に結果が表示される．点数は以下の式を用いて静的，動的テストを合わせて 10 点満点で計算される．

$$(\text{点数}) = \frac{(\text{採点項目とテストケースの正解数})}{(\text{採点項目とテストケースの合計数})} \times 10$$

この自動テストプログラムを利用することで，学生は自ら作成したプログラムの誤りを見つけ，修正することができる．

```
静的テスト開始
起動：OK
VBox：OK
Label：OK
TextField：OK
Button：OK
Label：OK
静的テスト成功
動的テスト開始
入力：12EC34
出力：NG
テストケース1成功
入力：23EC456
出力：OK
テストケース2成功
入力：34EC5678
出力：NG
テストケース3成功
入力：
出力：NG
テストケース4成功
【実行対象:CheckSIDApp，学籍番号:17EC000，名前:千住旭，点数:14】
```

図 10　自動テストの動作画面

5.4. 学生の課題自動採点の動作

5.3 のテストプログラムと XML ファイルを利用して，数百人の学生の課題をまとめて採点するシステムを開発した．学生のプログラムとテストプログラムを自動でコンパイルし，実行する手法である．この自動採点システムの動作画面を図 11 に示す．フォルダに格納されている学生のプログラムを順番にコンパイルし，テストを行っていく．ここでは学籍番号と名前，点数のみが結果として出力される．このシステムを利用することで数百人の学生のプログラムを素早く自動採点することができる．

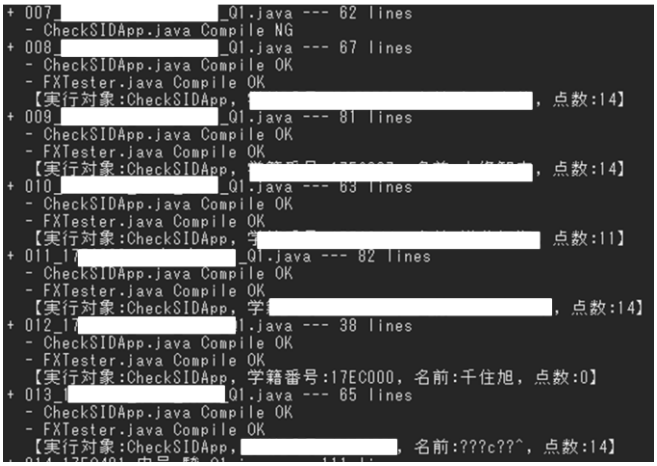

図 11　自動採点の動作画面

6. 評価

6.1. 実験データ

本大学の授業「オブジェクト指向プログラミング」で出題された演習課題を用いてシステムの評価を行う．2018 年度と 2019 年度の授業で出題された演習課題は，自動採点対象でないものを除くと全部で 11 種類ある．本論文で評価の対象とする 7 種類の演習課題とその内容について表 3 に示す．また，図 5 で示したアプリケーションを除く 6 つの課題のレイアウトを図 12 ~ 図 17 に示す．

表 3　実際の演習課題と提出者数

課題名	内容	提出者[人]
CheckSIDApp	学籍番号の形式が正しいか判定する	87
AddTaxApp	税込み価格を計算する	90
SplitBillApp	割り勘したときの価格を計算する	87
LunchCalcApp	選んだメニューの合計金額を計算する	95
DenominationApp	入力した金額の金種の枚数を表示する	104
RamenOrderApp	選んだラーメンの金額を計算する	101
TicketCalculator	選んだチケットの合計金額を計算する	93

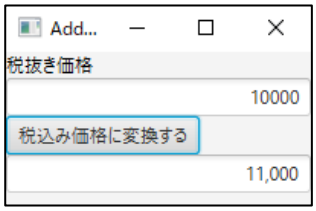

図 12　税込み価格計算アプリケーション

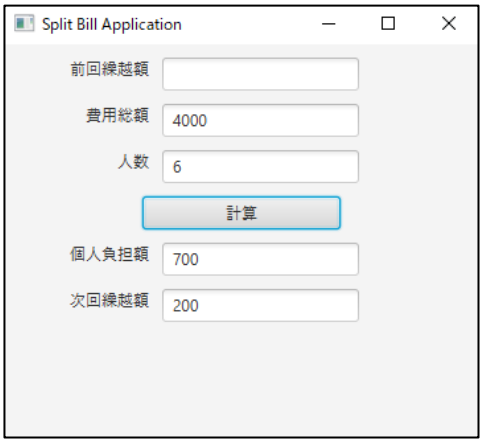

図 13　割り勘価格計算アプリケーション

図 14　ランチ価格計算アプリケーション

図 15　金種価格計算アプリケーション

図 16　ラーメン価格計算アプリケーション

図 17　チケット価格計算アプリケーション

6.2. 採点精度の評価

6.2.1. 静的テストの評価

我々が提案した自動採点システムが正しい精度で採点が行われているかを評価する．具体的には，手動での採点結果と提案手法の採点結果と比較して正しく採点されているか評価する．静的テストの採点項目数は全ての項目を対象とする．手動採点との採点精度を比較した結果を表 4 に示す．表 4 を見ると，ほとんどの演習課題で手動採点と同じ精度で採点することができていた．このことから，提案した手法では手動採点と比べて高い精度で静的テストが行えていることがわかる．しかし，TicketCalculator の自動採点において，誤った採点結果を示すことがあった．これは BorderPane の自動採点に問題があることが分かった．このアプリケーションでは BorderPane の上，中央，下にそれぞれコンポーネントが配置されているが，レイアウトを取得する際に異なった順番で取得されることがあった．その結果，静的テストの自動採点を正しく行うことが出来なかったと考えられる．今後は BorderPane を使用した GUI アプリケーションに対しても自動採点が行えるように改良する必要がある．

表 4　採点精度の比較結果

課題名	採点項目数	手動採点 [%]	提案手法 [%]
CheckSIDApp	6	100	100
AddTaxApp	5	100	100
SplitBillApp	18	100	100
DenominationApp	22	100	100
TicketCalculator	12	100	90.3

6.2.2. 動的テストの評価

次に動的テストにおける採点精度を評価する．動的テストでも同様に手動採点での結果と提案手法の結果を比較して評価を行う．それぞれの演習課題で動的テストを行った結果を表 5 に示す．表 5 を見ると，静的テストと同様にほとんどの演習課題で手動採点と同じ精度で採点できていることがわかる．

しかし，SplitBillApp の採点精度が 100%にならなかった．割り勘価格計算アプリケーションは費用総額か人数のどちらかが入力されていない状態でボタンを押すとエラーウィンドウ（アラート）が表示されるアプリケーションとなっている．しかし，学生のアプリケーションには費用総額と人数を入力しているにも関わらずエラーウィンドウが表示されるものがあった．このような誤ったアプリケーションの採点結果において一部のアプリケーションが満点と採点されているものがあった．この採点ミスが原因で採点精度が 100%にならなかった．今後はアラートが表示されるアプリケーションで正しく採点されない原因を見つけ，採点精度を上げる必要がある．

また，TicketCalculator の採点精度も 100%にならなかった．このアプリケーションではボタンを押すとコンソール上ではエラーメッセージが表示されるが，GUI 上では正しく動作しているものがいくつかあった．このようなアプリケーションを自動採点すると画面がフリーズしてしまった．つまり，手動採点ではエラーが表示されているにもかかわらず正しく動作し，自動採点ではプログラムの動作が停止してしまい採点できなかった．今後はこのようなエラーに対して正しく採点が行えるように改良する必要がある．

表 5　採点精度の比較結果

課題名	テストケースの数	手動採点 [%]	提案手法 [%]
CheckSIDApp	4	100	100
AddTaxApp	9	100	100
SplitBillApp	12	100	92.0
LunchCalcApp	10	100	100
DenominationApp	9	100	100
RamenOrderApp	10	100	100
TicketCalculator	9	100	92.5

6.3. 採点時間の評価

次に既存手法と比較したときの採点時間の評価を行う．ここで比較する既存手法は 2 章で紹介したスクリプトベースの自動採点システムとする．それぞれの手法に対して 5.4 節で示した自動採点プログラムを用いて採点を行い，main メソッドが実行されてから終了するまでの時間を計測する．2 つの手法で採点時間を計測した結果を表 6 に示す．2 つの手法を比べると，提案手法は既存の手法に比べて僅かに遅いことがわかる．これは提案手法では毎回 XML を読み込む処理があることから，既存手法よりわずかに時間がかかってしまうということが考えられる． しかし，100 名弱の課題採点で 6〜10 秒の差なので大きな問題ではないということが言える.それぞれの課題の採点時間を比べると，CheckSIDApp と AddTaxApp の採点時間に対して他の 5 つの課題では採点時間が増えている．このことから，採点項目数が多いほど採点時間が増えると考えられる．

表 6　採点時間の比較結果

課題名	既存手法 [s]	提案手法 [s]
CheckSIDApp	160.0	166.4
AddTaxApp	166.0	175.5
SplitBillApp		215.4
LunchCalcApp	211.0	217.3
DenominationApp	223.4	233.2
RamenOrderApp	226.6	238.1
TicketCalculator	228.5	215.0

6.4. 全体の考察

我々が開発した自動採点システムを手動採点の採点結果と比較したところ，静的テスト，動的テストともに高い精度で自動採点を行うことができた．しかし，静的テストでは正しく採点できない項目があり，動的テストでは誤ったアプリケーションに対して満点を与えてしまう問題が見つかった．実際の授業で運用した場合，このような採点ミスが起こってしまうと学生の評価が公平でなくなってしまうことから，自動採点の精度を 100%にする必要がある．

また，提出した学生全員分の課題採点にかかる時間を既存手法と比較した結果，提案手法は既存手法に比べてやや劣ったが，比較的同じ速さで自動採点を行うことができた．今後は既存手法より速く採点を行えるようにするために，使用する XML ファイルを減らす手法や自動採点するための処理の見直しを行う必要がある．

既存手法では自動採点したいアプリケーションに対して手作業でスクリプトを記述しなければならなかったが，提案手法では採点者が作成した模範解答から XML を生成して自動採点を行うので，事前準備にかか

る時間を大幅に減らすことができた．既存手法の自動採点用スクリプトは作成するのに 40 分～1 時間ほどかかるので，7 種類の GUI プログラミング課題を自動採点するには 7 時間ほどかかってしまう．それに対して，提案した手法は実際に GUI を動かしながら XML を生成することができるので，1 つのプログラミング課題に対して 5 分ほどで XML（テストケース）を生成することができる．つまり 7 種類の GUI プログラミング課題を自動採点する準備にかかった時間は，既存手法と比べて 6 時間以上短縮することができたといえる．しかし，既存手法は提案手法と比べて多くの種類のプログラミング演習の自動採点に対応しているため，今後は提案手法で自動採点することができる GUI プログラミング課題を増やすために機能を拡張していく必要がある．

7. まとめ

GUI プログラミング課題の自動採点において，模範解答を利用してレイアウト,イベントの情報を XML ファイルに格納するプログラムを作成したことにより，様々な GUI アプリケーションで簡単に自動採点を行うことができた．実際の演習課題を用いて学生の課題を自動採点した結果，手動採点と比べて高い精度で採点を行うことができた．採点にかかる時間は既存の手法よりやや劣ったものの，模範解答を用意するだけで自動採点ができることから，自動テスト用のプログラムを作成する時間を短縮することができた．しかし，まだ全ての演習課題を自動採点することができない．例として画面が遷移するものや，外部ファイルを読み込んで出力するようなアプリーションには対応していない．今後は採点精度を手動採点と完全に一致させるとともに自動採点に対応する GUI アプリケーションを増やすことが課題である．

参 考 文 献

[1] 小学校プログラミング教育の手引(第二版)，平成 30 年 11 月，文部科学省．

[2] Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon, “GUITAR: an innovative tool for automated testing of GUI-driven software”, Autom Softw Eng, pp. 65-105, 2014.

[3] Ushio Inoue, “GUI Testing for Introductory Object-Oriented Programming Exercises”, 5th International Conference on Computational Science/ Intelligence & Applied Informatics, pp. 1-13, 2018.

[4] Omar El Ariss, Dianxiang Xu, Santosh Dandey, Brad Vender, Phil McClean, and Brian Slator, “A Systematic Capture and Replay Strategy for Testing Complex GUI based Java Applications”, 2010 Seventh International Conference on Information Technology, pp. 1038-1043, 2010.

[5] R. M. L. M. Moreira, A. C. R. Paiva, and A. Memon, “A pattern-based approach for gui modeling and testing”, in Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on, pp. 288–297, 2013.

[6] Pedro Costa, Ana C.R. Paiva, and Miguel Nabuco, “Pattern Based GUI Testing for Mobile Applications”, 9th International Conference on the Quality of Information and Communications Technology, pp. 66-74, 2014.

[7] Rui Carvalho, “A Comparative Study of GUI Testing Aproaches”, Faculdade de Engenharia da Universidade do Porto, 2016.

[8] Snyder, J., Edwards, S.H., Perez-Quinones, “LIFT: taking GUI unit testing to newheights”, Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, pp. 643-648, 2011.

[9] Alex Ruiz, Yvonne Wang Price, “Test-Driven GUI Development with TestNG and Abbot”, IEEE Software, pp. 51-57, 2007.

[10] Claus Klammer and Rudolf Ramler, Heinz Stummer, “Harnessing Automated Test Case Generators for GUI Testing in Industry”, 42th Euromicro Conference on Software Engineering and Advanced Applications, pp. 227-234, 2016.

[11] 内藤広志，斉藤隆，水谷泰治，“GUI プログラミング課題の自動採点方式について”，情報処理学会研究報告，p81-88, 2008.

[12] 立石良生，井上潮，“GUI プログラミング課題自動採点方式の検討”，DEIM Forum 2019，E2-3.