N_002

Javaプログラミング初学者に対するテスト方法学習支援ツール

A Learning Support Tool for Testing Method of Java Programs

上河内　頌之[†]　　松浦　佐江子[‡]

Nobuyuki Kamigochi　Saeko Matsuura

1．はじめに

我々はJavaプログラミング初学者(以下学習者とする)に対するテスト学習支援の研究を行ってきた[1]。[1]では、定義する全メソッドの仕様とシグネチャを明記した課題を用いて、Javaプログラムを対象に学習者に対するテスト方法学習支援を提案した。本稿は提案する支援に基づいたテスト方法学習支援ツールの設計について述べる。本ツールでは学習者に効果的なテスト方法の学習を提供するため、学習者個々の理解の相違に着目し、学習者自身で各ステップの内容を変更可能とした。

2．テスト方法の学習支援

この章では、最初に提案するテスト方法学習支援の目的を示す。そして具体的な各ステップの支援内容を本学2年生のプログラミング演習課題を例にして説明する。

2．1　学習支援の概要と目的

学習者の問題点として『テスト方法が分からない』、すなわち『プログラムのどこから、どのように、どの程度行うのか分からない』がある。この問題に対し、まずテストプロセスを『テストするメソッドの決定・テストデータ作成・テストプログラム作成と実行』の3ステップで定義する。そして課題に対し学習者が作ったプログラムから生成する情報と教員が用意する情報を学習者に提供して選択や入力を行わせる。テストするメソッドの決定において『どこから』テストを行うべきかを、テストデータ作成・テストプログラム作成を行い、テスト結果を確認する事で『どのように』テストを行えばよいかを学習させる。さらにメソッド毎に用意した全てのテストケースと、全メソッドに対して上記のステップを繰り返す事で『どの程度』テストすればよいかを学習する。また誤った選択や入力の際にその根拠を示し、各ステップを正しく実行するまで作業を繰り返す事で、下記の作業を自力で行なえる事を本支援の目的とする。

- テストするメソッドを適切に決定出来る
- テストケースに沿った妥当なテストデータが作成出来る
- テストプログラムとは何かを理解し、作成出来る

2．2　用語の定義

本稿中の用語の定義を行う。**課題**とは定義するクラスの全フィールドの型と名前、全メソッドの自然言語で書かれた仕様とシグネチャが定義されたものである。**テストケース**とは自然言語で書かれたテスト仕様と同義で、テストケース名・そのテストケースに対応するテストデータ・期待する結果の3点から構成される。**テストプログラム**とはオブジェクトを作り、状態を設定してメソッドを実行し、結果確認を行うための簡易プログラムである。**テストデータ**は入力値(フィールド値・メソッド引数値)、期待する結果(返り値・フィールドの変更値)、コンストラクタ引数の3点で構成される。

†芝浦工業大学大学院,Shibaura Institute of Technology graduate school engineering research course

‡芝浦工業大学, Shibaura Institute of Technology

2．3　説明に用いる具体例

ここからは具体例を用いて、用意する情報、各ステップの支援内容を説明する。

具体例の課題は、長方形を表すRectangleクラスを継承した位置情報付き長方形のPlaceRectangleクラスを更に継承した、色の属性値を持つColoredPlaceRectangleクラスとそのメソッドを作成させるものである。図1に具体例のクラス図を示す。

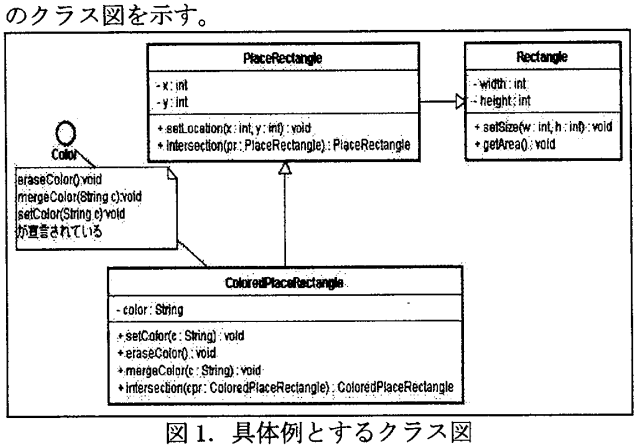

図1．具体例とするクラス図

本提案では、定義済みのクラスのメソッドも含め、全メソッドに対しテストを行う。

2．4　学習支援に必要な情報

本支援を行う上で学習者の作ったプログラムからクラス毎、メソッド毎に表1の情報を抽出する必要がある。

表1．プログラムから抽出する情報

[クラス毎に取得する情報]

A:クラス情報
　パッケージ名、スーパークラス名、インスタンスフィールド名と型、メソッド名とシグネチャ

[メソッド毎に取得する情報]

B:他メソッドの呼び出し情報
　他メソッド呼び出し個数と名前

C:条件分岐数
　if, switch, for, while 文の数

D:フィールド参照情報
　参照しているインスタンスフィールド数と名前

E:引数参照情報
　引数の個数と名前

F:結果情報
　インスタンスフィールドの代入文と return 文

また、教員からは表2の情報が必要となる。情報Iのコンストラクタ条件とは、コンストラクタ名と引数が満たす条件の2点から構成され、課題を分析して作成する。コンストラクタによりオブジェクトを表現するために適切な値を引数として与える必要がある事を示すために用いる。

表２．教員が用意するプログラム仕様情報

G:参照実装
　課題に対し、要求を満たすプログラム
H:参照テストケース
　メソッド毎の要求を満たすテストケース
I:コンストラクタ条件
　定義されているコンストラクタの具体的な引数の値の条件を自然言語で示したもので
J:具体的なテストデータ（１つ）
　１つのテストケースを満たすテストデータ

表１のA,B,Cをテストするメソッドの決定支援に、表１のD,E,Fと表２のH,Iをテストデータ作成支援に、そのテストデータと表２のGをテストプログラム作成と実行支援に用いる。また情報Jは各ステップの支援内容を学習者に理解させるため、支援を自動的に実行するチュートリアル中で用いる。

２．５ テストするメソッドの決定支援

このステップは以下の３項目で行う。

1. 表１の情報A,B,Cからメソッド毎にヒントを与える。
2. ヒントとメソッドのソースを見て、メソッドを決定。
3. 正しい選択をしたら、次のフェーズへ。誤った選択をしたら、誤った理由をヒントとして示し2. へ戻る。

ここのヒントは、学習者にメソッド決定の際に着目すべき情報を示すため、情報Aからパッケージ名、スーパークラス名、委譲している場合はそのクラス名、加えて情報Bの個数と名前,Cから条件分岐の個数を図２の形で表示する。この情報から特に呼び出し回数とクラス情報を読み取らせて決定を行わせる。テスト終了チェックとは、呼び出しているメソッドや、継承や委譲関係にあるクラスの全メソッドがテスト済みかを示すもので、×は未終了を示す。

本稿では学習者に対し、他のメソッドに及ぼす影響が少ないメソッドからテストするべき事を示すため、呼び出しているメソッドや委譲や継承しているクラスのメソッドが全てテスト済み、もしくは呼び出し回数が０で委譲や継承しているクラスが無い選択を適切な選択とする。またメソッド構造情報は、後のテストデータ作成支援に進んだ時、条件分岐が多いなどの複雑な構造だと考えるテストケースが増える事を認識させるため、正誤の判定には用いない。

PlaceRectangle ： intersection(PlaceRectangle)			
メソッド呼び出し情報			
呼び出し回数	1	テスト終了チェック	
呼び出しているメソッド	PlaceRectangle(int,int,int,int)	×	
メソッド構造情報			
if	2	for	0
switch	0	while	0
クラス情報			
パッケージ	Default		
親クラス	Rectangle		
親クラスのテストチェック	×		
委譲しているクラス	無し		

図２．メソッド決定支援のヒントの形式
(PlaceRectangle クラス intersection メソッド)

また不適切なメソッドを選択した場合、誤っている理由を学習者に把握させるため、情報A,B,テスト終了チェックから図３のように示す。示された情報を確認して、再度メソッドの選択を行い、適切なメソッドが選択されるまでこのステップを繰り返す。

選択しているメソッド	不適切である理由
PlaceRectangle ： intersection(PlaceRectangle)	PlaceRectangle(int, int, int, int)がテストされていない Rectangle クラスのメソッドを全てテストしていない

図３．誤った選択をした時の表示

今回はColoredPlaceRectangleクラスで定義した、下に示すmergeColorメソッドを正しく選択したとして後の説明を行う。このメソッドは、課題で指定された引数cと色の属性値colorの組み合わせにより、色の属性値を変更するメソッドである。

```
public void mergeColor(String c){
        if(c == DEFAULT_COLOR){
        }
        else if(color == DEFAULT_COLOR){
            color = c;
        }
        else if(c == "yellow" && this.color == "blue"
|| c == "blue" && this.color == "yellow"){
            color = "green";
        }
        else{
            color  = "white";
        }
    }
```

２．６ テストデータ作成支援

この支援を行うにあたり教員が用意する情報、参照テストケース(情報H)とコンストラクタ条件(情報I)が必要となる。このメソッドのH,Iを図４に示す。

テストケースNo	テストケース	期待する結果
1	色の属性値がyellow,引数値がblue	色の属性値がgreen
2	色の属性値がblue,引数値がyellow	色の属性値がgreen
3	引数値がDEFAULT_COLOR	変化無し
4	色の属性値がDEFAULT_COLOR	色の属性値が引数値
5	1～4以外の組み合わせ	色の属性値がwhite

コンストラクタ	引数が満たす条件
ColoredPlaceRectangle(int,int,int,int)	全て０以上の整数値

図４．テストケース・コンストラクタ条件

このステップは以下の２項目で行う。

1. 図４の情報からテストケースを示しヒントを与える。
2. テストケースを選択した後、データ作成を行う。

このヒントは、メソッドをテストするにはテストケースを考え、それに従うテストデータを作る事、また表示されたテストケースを全て行う事で十分テスト出来る事を学習者に示すため、図４のテストケースのみを表示しテストケースを選択させる。

そして学習者にテストデータとは入力値・期待する結果・コンストラクタ宣言時に与える引数値の３点から構成される事を示すため、{選択したテストケースとそれに対応する期待する結果、作成する変数の名前とその型、コンストラクタの引数が満たすべき条件}を図５のように示しながら、図６の形で入力を行わせる。

テストケースNo	テストケース	期待する結果
1	色の属性値がyellow,引数値がblue	色の属性値がgreen

コンストラクタ	引数が満たす条件
ColoredPlaceRectangle(int,int,int,int)	全て０以上の整数値

作成する変数	変数の型
color	String
c	String

図５．テストデータ作成支援のヒント

図6には記述済みのmergeColorメソッドのテストデータを示す。選択したテストケース、＜入力値＞のcolor,cの箇所は情報 D,E の名前から＜期待される結果＞の(無し),colorの箇所は情報Fから＜コンストラクタ＞には定義しているコンストラクタと引数を自動的に示し、学習者に[：]の右へ具体的なデータを入力させる。

```
テストケース：色の属性値がblue,引数値がyellow
＜入力値＞
  ＜参照するインスタンスフィールド＞
    color  :  blue
  ＜引数＞
    c      :  yellow

＜期待する結果＞
  ＜返り値＞
  (無し)
  ＜変更されるインスタンスフィールド＞
    color  :  green

＜コンストラクタ＞
    ColoredPlaceRectangle(int w,int h,int x,int y)
    w      : 10
    h      : 10
    x      :  0
    y      :  0
```

図6. テストデータ入力形式

２．７　テストプログラム作成と実行支援

このステップは以下の３項目で行われる。

1. テストプログラムのテンプレートを図７の形で表示。
2. テンプレートに従い、プログラムを作成する。
3. 完成したプログラムを実行し、結果を判断。

ここでは、学習者にテストプログラムとは何かを認識させるため、ヒントとしてテストプログラムのテンプレートを表示する。ここでは各フィールドのgetメソッドが定義されている場合のmergeColorメソッドのテストプログラムのテンプレートを図７に示す。クラス名はテスト対象メソッド名にTestを付け、作成したテストデータの＜入力値＞、＜コンストラクタ＞の情報をコメントで出力する。また結果確認にはassertを用いてテストデータの＜期待する結果＞の情報をコメントで出力し、学習者に入力してもらう。

```
public class TestMergeColor{
    public static void main(String[] args){
    //コンストラクタColoredPlaceRectangle(int w,int h,int x,int y)に
    //w = 10, h = 10, x = 0, y = 0としてオブジェクトcprを作成する。

    //colorの値を" blue" に設定

    //mergeColorを" yellow" を引数にして実行

    assert cpr.getColor() == /*期待結果*/ : "期待結果greenではない¥n
          実行結果="+cpr.getColor();
    System.out.println("期待結果と実行結果が一致しました");
    }
}
```

図7. テストプログラムテンプレート(mergeColor)

テンプレートに従い作成したテストプログラムをコンパイルし、学習者のプログラムに対して実行し結果を判断する。そしてテストの終了を意識させるため、正しい結果になれば、現在選択しているメソッドの残りのテストケースが無くなるまで、またそのメソッドの全テストケースが終了すればメソッド決定支援に戻る。こうして各ステップを用意した全テストケースが正しく終了まで繰り返す。

しかし、エラーが発生する場合がある。これは

①学習者が作成したプログラムが誤っている
②作成したテストデータが誤っている
③作成テストプログラムが誤っている

の３つの原因が考えられる。そこで表２の参照実装(情報G)を用いて作成したテストプログラム（以下testとする）を実行し原因の特定を行う。まず参照実装に対しtestを実行し期待通りの結果が得られれば①と原因が特定し、学習者の作った元のプログラムの修正を指示する。参照実装に対しtestを実行して誤った場合、まずテストデータが正しいと仮定し、図８のようにコメントに従い入力した箇所に下線を、また自分が作ったテストデータを太字で強調表示しテストデータとの比較を行い、テストプログラムを修正する。テストデータ通り修正しても誤っている場合、原因が②と特定しテストデータ作成ステップへ戻ってデータを作り直させる。

```
public class TestMergeColor{
    public static void main(String[] args){
        ColoredPlaceRectangle cpr =
            new ColoredPlaceRectangle(10,10,0,0);
        // ColoredPlaceRectangle(w,h,x,y) w=10,h=10,x=0,y=0
        cpr.setColor("blue");
        // color : blue
        cpr.mergeColor("green");
        // c : yellow
        assert cpr.getColor == "green" : "期待結果と違います¥n実行結果
"+cpr.getColor();
        //期待結果 : green
        System.out.println("実行結果と期待結果が一致しました");
    }
}
```

図8. 強調表示したテストプログラム

このように一つのテストプログラムが自分のプログラムに対して正常に動作するまで繰り返す。

３．テスト方法学習支援ツール

本提案をツールとして実現する際、学習者の苦手な分野を集中的に学習させるために、学習者の理解状況を把握出来る新たな学習者モデルの研究を行なっている[2]でも述べられているように、学習者の理解状況を把握する必要があり、考慮する必要がある。そのため本ツールは学習者の理解度により各ステップの支援内容を学習者自身で変更出来るようにして、学習支援を行う。本ツールの実行画面を図９に示す。また本ツールを用いたテスト方法の学習は以下のように行なわれる。

(1)学習者が作成したプログラムを入力する。
(2)提案したテスト方法学習に従いテストを行う。
(3)任意のステップが終われば学習者自身の理解に応じてステップ毎に理解度を変更する。

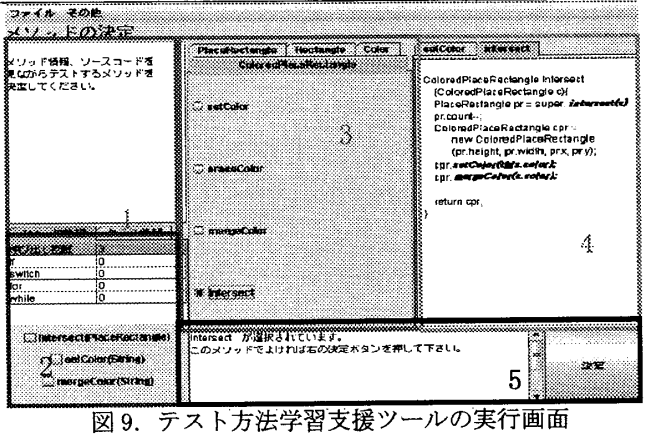

図9. テスト方法学習支援ツールの実行画面

３．１　ツールの概要

表３に本ツールの画面構成要素を示す。また図９の数字は表３と対応している。

表3．ツール画面構成要素

1：指導画面 現在のステップでの作業内容の説明文を示す 2：ヒント画面 現在のステップでのヒント情報を示す 3：作業画面 各ステップでの作業を行う 4：該当コード画面 現在対象としているソースコードが表示される 5：作業内容確認画面 3での作業内容を表示する

1で説明を行い、2，4に表示した情報を見ながら3で作業を行い、作業結果を5に表示する。ヒント情報とはそれぞれのステップで図2、5、6に該当するもの、作業とは各ステップでのメソッドの選択、テストデータ作成、テストプログラムテンプレートに従って作成する事、を指す。

3．2　学習者に応じた支援

ツールとして学習支援を提供する時、学習者によってテストプログラムの形式が分からない等学習が困難となる箇所が異なるため、学習者の理解度に応じて支援内容を変更する事が必要である。本稿では、各ステップでの支援内容を、学習者が自分の理解に応じて変更可能とした。各ステップの作業の終了時に、理解度を変更し、次回から違う支援内容で各ステップが行える。

本稿での理解度による各ステップの支援内容は表4のように定義した。理解度0では、支援内容を理解させるため、支援内容を自動実行するチュートリアルを行う。理解度1は、そのステップの流れは知っているが、行う内容をよく理解していない状態を想定している。そのためヒント情報に対応する変数やメソッド呼び出しなどを太字で表示したソースとヒントを用いてそのステップを行う。理解度2は、内容をある程度理解出来たと学習者が判断した状態を想定している。そのため、提供するヒントと太字表記していないソースを用いてそのステップを行う。また各ステップの内容の理解には影響無いと考えたため、表示するヒントは変更しない。

表4．理解度による支援内容の違い

メソッド決定支援	
理解度	支援内容
0	チュートリアル実行
1	ソースで着目する箇所（インスタンスフィールド、メソッド呼び出し）を太字表示
2	ソース上で特に表示を行なわない
テストデータ作成支援	
理解度	支援内容
0	チュートリアル実行
1	作るデータの変数表示、コードでの変数の太字表示
2	作るデータの変数表示、コードでの太字表示無し
テストプログラム作成と実行支援	
理解度	支援内容
0	チュートリアル実行
1	どこに何を作るかを示したテンプレート表示
2	クラス名、main文宣言のみ示したテンプレート表示

また、各ステップの内容の理解が細かく行えるように、チュートリアルの内容を表5のように定義した。表中にある説明とは、例えばメソッド決定支援では『メソッド決定にはメソッド呼び出し回数に着目して行う。』等文章を表示して行う。

表5．チュートリアルの内容

テストするメソッドの決定支援	1．何(メソッド呼び出しとクラス依存)に着目するかを説明
	2．誤った選択を自動的に行う
	3．誤った根拠を図3のように示す
	4．正しいメソッドを自動的に選択する
テストデータ作成支援	1．テストケースからテストデータを作成する事を説明
	2．選択しているメソッドの何に着目してデータを作るかを説明
	3．情報Jと対応するテストケースを選択し、テストデータを示す
テストプログラム作成と実行支援	1．テストプログラムとは何かを説明
	2．そのメソッドのテストプログラムテンプレートを図7のように示す
	3．インスタンスを生成し、状態を設定する文を自動的に生成
	4．メソッドを実行する文を自動的に生成
	5．結果判定文を生成
	6．出力判定の見方の説明

また、例としてテストプログラム作成支援における理解度による情報量の違いを図10に示す。

理解度：1

```
public class TestMergeColor{
    public static void main(String[] args){
    //コンストラクタColoredPlaceRectangle(int w,int h,int x,int y)に
    //w = 10, h = 10, x = 0, y = 0としてオブジェクトcprを作成する。

    //colorの値を" blue" に設定

    //mergeColorを" yellow" を引数にして実行

    assert cpr.getColor() == /*期待結果*/ : "期待結果greenではない¥n
          実行結果="+cpr.getColor();
    System.out.println("期待結果と実行結果が一致しました");
    }
}
```

理解度：2

```
public class TestMergeColor{
    public static void main(String[] args){

    }
}
```

図10．理解度での情報の変化(テストプログラム作成支援)

4．おわりに

本提案では、定義するクラスで定義するフィールド、全メソッドのシグネチャ・仕様が規定された課題を前提としている。そのため、フィールドの名前が違う場合に対しては、テストケースに対するテストデータを用意する事が出来ない。しかし、名前が違っていても宣言するフィールドの意味は同一のはずである。そのため、これらのフィールドの意味を把握する事が出来れば、違った名前であっても教員がそれを気にする事無くテストケースを用意する事が出来るはずである。そして、提案した流れによりテストデータを作成する事が可能となるのではないかと考える。

本稿では、理解度による支援内容を表4のように定義して、それぞれの支援を行っている。そのためこの内容が妥当かどうかを検討する必要がある。今後、ツールのインタフェースを洗練し、本学の生徒に対してツールを適用しこの方法により、テスト方法を理解する事が出来たかどうかの評価を行う予定である。

[参考文献]

[1] 上河内頌之, 松浦佐江子：Javaプログラミング初学者に対するテスト方法の対話的な学習支援, 情報処理学会全国大会第68回, 6K-6, 2006

[2] 砂長祐、桜井将人、山本洋介、古宮誠一：実験による拡張オーバレイモデルの有効性確認, 情報処理学会全国大会第68回, 3H-3, 2006