プログラムの抽象度を用いたコーディング過程把握手法の提案

2009SE267 鈴木那由多　2009SE276 高山直　2009SE284 寺尾勇紀
指導教員：蜂巣吉成

1　はじめに

情報系の大学では，学習者が講義中に演習という形式で，PC でプログラムを記述し，プログラミングを学ぶという方法が広くとられている．学習効果をあげるために，教員が学習者の進捗状況を把握し，指導することは重要である．しかし，次の理由などから，進捗状況を十分に把握できているとは言えない．

1. 学習者が質問をすることに抵抗を感じ，質問をしない．
2. 学習者が自分のプログラムの問題箇所を把握するのに時間がかかる場合があり，適切な質問ができない．

教員が教室を巡回することで，学習者の進捗状況をある程度把握することは可能であるが，講義時間には限りがあるので，各学習者に割ける時間が限られてしまう．TA を活用した場合，進捗状況が TA 毎に把握され全体として集約できない．さらに，短時間では把握できない問題があったり，同じような指導内容であるにも関わらず，学習者毎に個別に対応するような状況が起こっては，効率が悪い．教員が巡回を行う方法以外にも指導が必要な学習者を発見でき，多くの学習者が試行錯誤している箇所を容易に把握することができれば，学習者個人や全体にヒントを与えるなどの指導を行い，効率的に学習効果をあげることができる．

学習者の進捗状況を把握することで学習効果を効率的にあげることを目的として，教育支援システムがいくつか提案されている．[1]，[2] では，ソースコード編集履歴やコンパイル・実行結果，エラー情報などを基に，[3] では，メソッド呼び出し順の記録を基に分析を行っている．しかしこれらの方法では，提出されたプログラムを対象としているのでリアルタイムに指導できない．また，模範解答と比較することでソースコードがどの程度似ているかを把握しているが，実際に学習者がどのような記述をしているかがわからず，進捗状況の把握が不十分である．

学習者が記述する字句や文，文の構造と時間によるそれらの変化を，教員が個人と全体共に把握し，指導を行うべきか否かを，その指導内容を含めて判断できるようになれば，進捗状況を把握できたと言える．そのためには提出されたプログラムやコンパイル可能なプログラムだけでなく，記述途中のプログラムを自動的に取得し，それを基に内容を分析する必要がある．

そこで本研究では学習者のコーディング過程把握に必要なソースコードを取得し，字句や文，文の構造を用いた進捗状況を分析する手法を提案する．Web 上に Web ベースの統合開発環境 (Web Integrated Development Environment:WebIDE) を構築して学習者から自動的にソースコードを取得し，それを基に整理・分析を行い，教員に学習者の進捗状況を通知することで，個人や全体の進捗状況を把握できるような支援を行うことを考える．問題の出題にあたり，教員は模範解答と学習者に提示する雛形を作成する．一般に用いられている Eclipse などの開発環境や既存のエディタなどでもプログラムの自動取得は可能であるが，そのための環境構築や更新作業などが必要である．初学習者にとってこの作業は敷居が高くなることが多く，また限られた講義時間を割いて行うことは，プログラミング学習の本質から外れてしまう．WebIDE を用いることにより，これらの手間が省けるという利点が挙げられる．

本研究における課題を挙げる．

- ソースコードをどのような単位で分析するのか
- 学習者による記述の違いをどう扱うのか
- 模範解答とどのようにして比較するのか
- 模範解答と異なった記述をどのように扱うのか

本研究では，模範解答や取得した学習者のソースコードを，言語処理系の基本構造に従い，字句解析・構文解析・意味解析のそれぞれの抽象度についての解析手法を参考に字句，文，文同士の関係の抽象度で分析する．それぞれの抽象度について，模範解答や学習者同士で比較した結果を教員に通知することで，学習者個人や，全体の傾向を効率よく把握できるようになり，学習者を指導することができる．

なお，本研究では，対象言語として，学習や開発に広く用いられている Java を用いる．ただし，サーバ上に開発環境を構築できれば，Java 以外も扱うことができる．

2　関連研究

学習者の進捗状況を把握する教育支援機能はいくつか提案されている．[1] ではコーディング，コンパイル，実行，提出といった過程を記録し，課題の進捗や受講者の相対的な進捗の悪さを可視化して講師に提示する．[3] ではメソッド呼び出し順の記録から試行錯誤の過程を分析し，学生の習得項目を算出する．[2] ではコンパイルによって検出されない誤りの検出を行っている．これらの研究では，情報を課題提出後に取得していたり，模範解答に近づいている学習者や課題に対して試行錯誤している学習者の特定はできるが，内容までを把握することができない．また，コンパイル後のプログラムが対象なので，コーディング途中で試行錯誤している場合は対処ができない．

3　要求分析

教員のプログラミング演習時におけるプロセスの現状を図 1 に示す．現状では，試行錯誤していたり質問をする学習者に対して，教員はその都度学習者のソースコードを確認し内容を把握しているので，指導が必要かどうかの判断に時間がかかる．また，同じような誤りを記述している学習者が多数いた場合は，効率的な指導のために一斉指導を行うが，多くの学習者の進捗状況を把握し

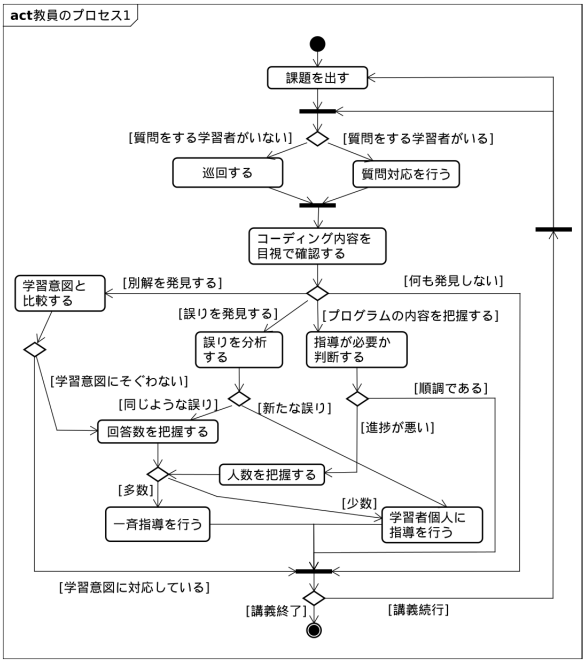

図 1 教員のプロセス

なければその判断ができない．

進捗状況を把握するためには次のことが必要である．

1. 模範解答と同様の記述をしている学習者について
   - どの記述が模範解答と同様であるか
   - 記述箇所はプログラム全体のどの程度か
   - 学習者が全体にどの程度いるか
2. 模範解答と異なる記述をしている学習者について
   - どのような記述をしているか
   - 学習者が全体にどの程度いるか

1 を分析することにより，図 1 における順調であるか進捗が悪い学習者がどの程度いるのかが把握できる．2 を分析することにより，図 1 における学習者がどのような誤りを記述し，その記述が全体でどの程度あるのかが把握できる．以上より一斉指導を行うのかどうかを判断するというプロセスを巡回以外でも行うことができる．

3.1 模範解答との比較

模範解答を分析し，その結果を学習者の分析結果と比較することで，模範解答にどの程度近いかを把握できる．模範解答と異なっていた場合は，以下に示すどの異なり方なのかを教員が判断し，多く記述される誤りについて指導することで，学習効果を効率的にあげることができる．

- 想定内の誤った記述・別解
- 想定外の誤った記述・別解

異なっていたソースコードが想定内なのか想定外なのかは，教員によって違うので，その判断は教員が行う．

3.2 抽象度を考慮した分析

個人の場合は模範解答と，全体の場合は学習者の集計結果と模範解答を比較し，正しい記述をしているかどうかを把握する．その場合，単にソースコード同士の差分を求めるだけでは，全く同じ記述でなければ正しいと判断することができない．同じ処理を記述していたとしても，字句や構文の違いから，少しずつ違うソースコードが記述される．この違いを吸収する必要がある．そこで，ソースコードを意味のあるまとまり毎に分割し比較することで，似た記述を同じ記述としてみなす事を考える．

4 演習における学習者の進捗状況把握方法

図 2 の出題例と図 3 の模範解答を用いて，提案する分析方法について説明する．出題形式については，教員が雛形と模範解答をあらかじめ用意しておくことを前提とする．また，雛形には教員が学習意図を考慮してクラスやメソッドのシグネチャ，変数などを定義しておき，学習者は残りのプログラムを記述する．

雛形

```
/** 与えられた雛形を参考に、指定した言語で
 *  月から季節の文字列を返す関数
 *  month2strを完成させなさい。
 * ただし、月から季節の文字列に変換する部分には
 * switch文を用いなさい。
 *
 *   lang   month   返り値
 *  ---------------------
 *   "Ja"  3, 4, 5   "春"
 *   "Ja"  6, 7, 8   "夏"
 *   "Ja"  9,10,11   "秋"
 *   "Ja" 12, 1, 2   "冬"
 *   "Ja"  上記以外   "エラー"
 *   "En"  3, 4, 5   "Spring"
 *   "En"  6, 7, 8   "Summer"
 *   "En"  9,10,11   "Autumn"
 *   "En" 12, 1, 2   "Winter"
 *   "En"  上記以外   "Error"
 */

public class Test {

    /** 月の判定をする */
 public static String month2str(int month,
                                String lang) {
      /** ここに処理を書く */
 }

 public static void main(String ars[]) {
    int mon;
    mon = 1;
    System.out.println(mon + " 月は "
                + month2str(mon, "Ja"));

    mon = 5;
    System.out.println(mon + " 月は "
                + month2str(mon, "En"));

  }
}
```

模範解答

```
public class Test {
    /** 月の判定をする */
  public static String month2str(int month,
                                 String lang) {
    String season = null;
      if (lang.equals("Ja")) {
        switch (month) {
         case 3: case 4: case 5:
           season = "春";
           break;
         case 6: case 7: case 8:
           season = "夏";
           break;
         case 9: case 10: case 11:
           season = "秋";
           break;
         case 12: case 1: case 2:
           season = "冬";
           break;
         default:
           season = "エラー";
        }
      }
      else if (lang.equals("En")) {
        switch (month) {
         case 3: case 4: case 5:
           season = "Spring";
           break;
             ・・・
        }
      }
    return season;
  }
                    ・・・
}
```

図 2 出題例

4.1 分析方法

本研究では，言語処理系の基本構造に従い，字句解析・構文解析・意味解析の，それぞれの抽象度についての解析手法を参考に，教員が設定した模範解答と学習者のソースコードに対して字句，文，文同士の関係の三つの抽象度で分析することで学習者の記述方法の違いを吸収する．また，その出現回数と記述人数を調べることにより，学習者全体でどのような記述があるかを把握する．

4.2 字句について

ソースコード内の予約語，修飾子，演算子，リテラル，雛形の変数に対して分析を行うことで，学習者が模範解答に現れる字句を正しく記述できているかが把握できる．ただし，学習意図には関係のない中括弧，カンマ，ピリオドや学習者が定義したクラス名，メソッド名，変数を分析の対象としない．

別解

```
public class Test {
    /** 月の判定をする */
    public static String month2str(int month,
                                   String lang) {
        switch(month) {
            case 3: case 4: case 5:
                if(lang.equals("Ja"))
                    return "春";
                else if(lang.equals("En"))
                    return "Spring";
            case 6: case 7: case 8:
                if(lang.equals("Ja"))
                    return "夏";
                else if(lang.equals("En"))
                    return "Summer";
            case 9: case 10: case 11:
                if(lang.equals("Ja"))
                    return "秋";
                else if(lang.equals("En"))
                    return "Autumn";
            case 12: case 1: case 2:
                if(lang.equals("Ja"))
                    return "冬";
                else if(lang.equals("En"))
                    return "Winter";
            default:
                if(lang.equals("Ja"))
                    return "エラー";
                else if(lang.equals("En"))
                    return "Error";
        }
        return season;
    }
    public static void main(String args[]){
                ・・・
    }
}
```

誤り解答

```
public class Test {
    /** 月の判定をする */
    public static String month2str(int month,
                                   String lang) {
        String season = null;
        if (lang == "ja") {
            switch (month) {
                case 3: case 4: case 5:
                    season = "春";
                    break;
                case 6: case 7: case 8:
                    season = "夏";
                    break;
                case 9: case 10: case 11:
                    season = "秋";
                    break;
                case 12: case 1: case 2:
                    season = "冬";
                    break
                default:
                    season = "エラー";
            }
        }
        switch (month) {
            case 3: case 4: case 5:
                season = "春";
                break;
            case 6: case 7: case 8:
                season = "夏";
                break;
            case 9: case 10: case 11:
                season = "秋";
                break;
            case 12: case 1: case 2:
                season = "冬";
                break
            default:
                season = "エラー";
        }
        return season;
    }
    public static void main(String args[]){
                ・・・
    }
}
```

図 3 想定される解答例

この分析方法では，模範解答と学習者のソースコードから抽出した分析対象の字句を比較することにより，模範解答には含まれない字句を抽出することができるので，学習者の字句の記述誤りや模範解答とは異なる字句を用いて学習を進めていることが把握できる．例えば，模範解答にはない “ja” や “spring” などの誤った文字列が増えたことを読み取れば，学習者が字句を誤って記述していることが把握できる．また，図 3 の別解と模範解答を比較すると，別解では模範解答に比べて “return” の出現回数が多いので，その記述が増えていくことを読み取れば，模範解答とは異なる方法で学習を進めていることが把握できる．しかし，字句の集計結果だけでは if 文などの条件式において誤った記述をしているのか，または記述自体をしていないのかを把握することができない．

4.3 文について

ソースコード内の式文，宣言文，選択文，繰返し文，ジャンプ文，ガード文に対して分析を行うことで，学習者が模範解答に現れる文を正しく記述しているかが把握でき，字句の抽象度で分析する際に生じる問題を解決することができる．また，文の構造は同じでもリテラルの記述が違うと異なる文として分析してしまうので，リテラルは対象としない．

この分析方法では制御文の条件式などがまとめられているので字句の抽象度では把握できなかった，制御文の条件式などをどのように記述をしているのか，または記述自体をしていないのかを把握することができる．例えば，図 3 の誤り解答では “if(lang ==)” という記述が増えたことを読み取れば，if 文の条件式において誤った方法で文字列の比較を行っていることが把握できる．しかし，文毎の集計結果だけでは，文の構造がどうなっているかが分からないので，どのような連接関係や制御依存関係で文を記述しているかを把握することができない．

4.4 文同士の関係について

文の抽象度で分析した後，文同士の連接関係と親子関係を分析する．それらを分析することにより，学習者が模範解答と同じ構造で文を記述しているかが把握できる．親子関係を分析するにあたり，親子だけでなく子孫までを考慮すると学習者間の違いが多くなり，違いを吸収できなくなるので親子関係の場合は二つずつ，また連接関係の場合は文の順序を把握するために全てをひとまとまりとする．例えば図 3 の別解では模範解答にはない制御依存関係である，“case” の子として “if(lang.equals())” が記述されたことを読み取れば，模範解答とは異なる方法で学習を進めていることが把握できる．また，誤り解答では “if(lang ==)” と連接関係にある “switch(month)” が記述されたことを読み取れば，誤った構造で文を記述していることが把握できる．

5 実装と検証

5.1 WebIDE の設計と実現

学習者のコーディング過程を把握するために，WebIDE を設計，実現した．本研究では 30 秒毎と WebIDE 上のボタンをクリックしたときにソースプログラムを取得する．次に実現した機能を示す．

- コンパイル・実行機能
- アカウント・ファイル管理機能
- ソースコード分析機能

本研究では，Java を対象としているので，WebIDE 内のプログラムで Java のコンパイルや実行が容易な，JavaServlet を用いて実現した．ファイルに対して行うことができる操作として，開く・作成・保存・削除がある．ディレクトリに対しては，作成・削除がある．またその他の操作として，ファイル名をクリックするとそれぞれに対応したメニューを表示できる．

標準入力値やコマンドライン引数については，プログラムの実行中に入力待ちを行うことができないので，予めその値を入力しておく．Web 上でのコンパイル方法については，保存されたソースコードをサーバ上で CompilerAPI[4] を用いてコンパイルし，外部にクラスファイルを出力するとともに，実行を行っている．Web ページ用の HTML，JSP ファイルは 129，108 行，JavaScript ファイルは 559 行である．WebIDE の基本機能である Java ファイルは 2487 行である．

5.2 実験・検証

本研究で提案した WebIDE を大学生 10 名に使用してもらい，進捗状況把握方法の検証を行った．図 2 の問題を実際に解いてもらい，学習開始から 5 分後，10 分後，15 分後の各学習者のソースコードを分析し，また検証方法として，分析結果と学習者のアンケートを照らし合わせ，把握できたことを抽象度毎にまとめた．

5.2.1 字句の比較

図 4 より，時間が経過しても模範解答にある文字列の比較で使われる equals メソッドが字句の集計結果にあらわれなかったことから，学習者は文字列の比較を行う方法を考えている，または，別の方法で文字列の比較を行っている場合が考えられる．また，雛形の main メソッドで出力を行っているにも関わらず，3 名の学習者の字句の集計結果から，“System”，“out”，“println” の字句が読み取れた．よって，季節の判定をするメソッド内でも余分な出力を行っていることが把握できた．

5.2.2 文の比較

字句の分析では，文字列の比較を行う方法を考えているのか，別の方法で行っているのかを判断できなかったが，図 5 より，文の比較を行い制御文内に言語を判定する変数 lang があったことから，別の方法で文字列の比較を行っていると判断した．また，時間の経過につれ，文字列の比較を if 文で行っていたのが，switch 文へ移行していることから，文字列の比較で試行錯誤していると判断した．アンケートよりその判断が正しかったことを確認した．

5.2.3 文同士の関係の比較

この問題では，二つの制御文が親子関係にある必要がある．文同士の関係を調べた結果，模範解答とは違う構造として，if 文と switch 文を連接して記述している学習者と二重 switch 文を記述している学習者を把握できた．

6 考察

アンケートによると，学習者が試行錯誤した箇所は，文字列の比較と返り値の変数の初期化であった．文字列の比較を学習者が試行錯誤していることは把握できたが，変数の初期化を把握することはできなかった，その原因は，雛形の設定にあり，雛形以外の変数は文毎の比較の際，各学習者が違う名前で変数を設定しても対応できるように消してしまう．今回の問題では雛形に返り値の変数を設定していなかった．今回のように判定結果や計算結果を返すメソッドを作成する問題では，結果を保存する変数を雛形に記述しておけば，より詳細な進捗状況の把握ができると考えられる．また，連接関係の集計結果について，今回の問題のように文が複数連接し，かつその数にばらつきが生じるような場合は，有効な結果は得られなかったので，分析方法を考え直す必要がある．

図 4 や図 5 に示した表だけでは進捗状況を直感的に判断することが難しい．模範解答と同様の記述と異なる記述を色分けして表示したり，全員が模範解答と同じ記述をした場合を 100 %として，分析時点での模範解答と同じ記述をした割合を表示することで直感的な判断を支援することができる．今回は一つの模範解答と学習者のプログラムを比較したが，考えられる別解も模範解答として比較したり，予め想定される誤りも同様な方法で比較することで，想定内の記述と想定外の記述を区別して表示させることもできる．

5分後

字句	回数	人数
if	9	6
lang	14	10
==	9	6
“ja”	1	1
“Ja”	6	5
switch	15	10
month	11	7
⋮		
System	1	1
out	1	1
println	1	1
season	23	2
S	1	1
epuals	2	1

10分後

字句	回数	人数
if	8	6
lang	13	10
==	14	4
“ja”	2	2
“Ja”	6	6
switch	23	10
month	20	8
⋮		
System	48	2
out	48	2
print	22	2
println	26	2
equals	2	1

15分後

字句	回数	人数
String	5	5
=	27	3
“null”	2	2
if	4	3
switch	16	7
“ja”	2	2
“Ja”	6	6
⋮		
System	40	3
out	40	3
println	40	3
print	10	1
equals	2	1

図 4 字句の集計結果

5分後

文	回数	人数
if (lang ==)	7	5
switch (lang)	4	3
switch(lang==)	1	1
if(lang.equals())	1	1
switch (month)	9	6

10分後

文	回数	人数
if (lang ==)	5	4
switch (lang)	5	5
switch(month)	14	9
if(lang.equals())	2	1

15分後

文	回数	人数
if (lang ==)	2	2
switch (lang)	11	7
switch(month)	14	7
if(lang.equals())	2	1

図 5 文の集計結果

7 おわりに

本研究では，学習者の進捗状況を字句，文，文同士の関係の三つの抽象度で分析する方法を提案した．既存の研究では困難だった学習者のコーディング過程を把握しつつ，悩んでいる箇所を特定することを目的とした．複数の学習者のソースコードを取得，分析するために，本研究では WebIDE を構築した．学習者のソースコードを実際に提案した方法で検証を行ったところ，学習者の悩んでる箇所二つの内一つを把握できた．

今後の課題として，提案した方法を WebIDE 上で実現する必要がある．また，集計結果を数値化し，教員が視覚的に進捗状況を把握できることおよび，より詳細な文同士の関係に対する対処，さらに，提案した手法を今回とは別の問題で用いた場合に，有効な結果が得られるかどうかを検証することも今後の課題である．

参考文献

[1] 齊藤俊, 山田誠, 井垣宏, 楠本真二, 井上亮文, 星徹, “プログラミング演習における受講生支援のためのコーディング過程可視化システム,” 信学技報, vol. 111, no. 481, SS2011-67, pp. 61-66, 2012.

[2] 櫻井桂一, “プログラミング実習を支援するための C プログラム誤り検出システムの開発,” 日本教育工学雑誌 25 ,pp. 67-70, 2001.

[3] 谷川紘平, ディンドンフォン, 原田史子, 島川博光, “C 言語関数呼出しの記録を用いた演習過程での習得項目の把握,” 電子情報通信学会論文誌, vol.J95-D, no.12, pp.2079-2089, 2012.

[4] Oracle Corporation and/or its affiliates, “*Community Development of Java Technology Specifications*,” http://jcp.org/en/jsr/detail?id=199, 2012.