社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
ET2005-31 (2005-09)

評価方法に従った自動採点を可能にする プログラム採点支援ツールの開発

藤原　巧† 　松浦　佐江子‡

† 芝浦工業大学大学院工学研究科　‡ 芝浦工業大学システム工学部

† ‡〒３３７－８５７０　埼玉県さいたま市見沼区深作 307

E-mail:　† m104089@sic.shibaura-it.ac.jp,　‡ matsuura@se.shibaura-it.ac.jp

あらまし プログラミングレポートの採点は様々な評価を行う必要がある。しかし採点作業は単に評価作業を行うだけでなく、ソースの表示やコマンドの入力、採点データの管理等の作業もあり、採点者に負担を与える。我々は採点作業の負荷軽減のためのツール開発に取り組んでおり、本稿ではそのツールの特徴と有効性について述べる。

キーワード　プログラミング授業、レポート採点、採点環境、自動化コンポーネント

Development of Programming Grading Supporting Tool for Automated Grading Components

Takumi Fujiwara†　Saeko Matsuura‡

† Shibaura Institute of Technology graduate school engineering research course

‡ Shibaura Institute of Technology system Department of Engineering

† ‡ 307 Fukasaku, Minuma-ku, Saitama-si, Saitama, 337-8570 Japan

E-mail:　† m104089@sic.shibaura-it.ac.jp,　‡ matsuura@se.shibaura-it.ac.jp

Abstract　The marker evaluates the programming reports by a lot of subjects. He should not only evaluate but also works a lot by display of the source and management of the grading data, etc. Therefore, grading reports gives a heavy work load to him. We develop programming grading supporting tool for automated grading components. This paper describes the features of this tool and discusses the trial results.

Keyword　Programming class, Report grading, Grading environment, Automation components

1. はじめに

プログラミングレポートの採点では数十～百数十のレポートに対してコンパイル、プログラムの実行結果の評価や記述されているプログラムの内容の元にアルゴリズムや構造の評価を行う作業だけでなく、ソースの表示、コマンドラインでのカレントディレクトリの移動、コンパイル、実行コマンドの入力といった評価作業のための前処理を繰り返し行う必要がある。更に採点結果は別途管理されているので、評価を行う場合には学生を間違えないように注意する必要がある。このように採点者は評価作業の他にも、前処理やデータ管理といった作業を行う必要がある。これらの問題を解決し採点を効率よく行うには、採点者が採点を行い易い環境を与えかつ評価作業の支援を行うことが必要になる。本稿ではプログラムレポート採点を支援するためのツールの開発を行い、採点支援ツールの機能とその有効性について述べる。

2. 採点作業の流れとその問題点

2.1. レポート提出・管理支援システムによるレポート提出

本学のプログラミングの授業では、レポート提出・管理支

援システム：C.S.E(Class Support Environment)[1,2]を用いて、レポートの電子データでの提出、一括管理を行っている。C.S.E では学生がフォルダ構成を定義して複数ファイルを提出することができるので、Java 言語でも学生の定義したファイル構成をそのまま提出することが可能である[1]。なお、C.S.E にはソースファイル以外に実行結果や考察等が提出される。C.S.E に提出されたソースコードを採点するにはコンパイル、実行を行う必要があるが、C.S.E は本研究室のサーバーで稼動しており、動作が保証されていないプログラムをサーバー上で実行することはシステム運営を妨げる危険を伴う行為である。採点作業はコンパイルや実行操作を必要とするのでサーバー上ではなく、ローカルな環境で行うことが望ましい。ローカルな環境は採点者がプログラムやテストを実行するために独自のライブラリやツールを使用する必要がある場合でもサーバー側より有効である。

2.2. 採点作業の流れ

C.S.E に提出されたレポートはファイル構成を維持したまま一括してダウンロードすることができる。採点はこのデータを用いて以下(1)～(3)の流れで評価を行い、全レポートを評価した後に(4),(5)の作業を行う。レポートは複数の設問で構成されることが多く、レポート数×設問数の数だけ(1)～(3)を繰り返し行う必要がある。

(1) レポートが入っているフォルダを開き、フォルダ内の全ソースファイルを表示する。ファイル構成が作成されている場合はフォルダを移動しファイルを開く必要がある。

(2) コマンドプロンプトを開き、カレントフォルダを移動する。コンパイルコマンド、実行コマンドを入力、実行し、結果を確認、記録する。プログラムの実行は多くの場合、異なる入力値によって何度も行われる。

(3) 言語の特徴を活かした実装がされているか、アルゴリズムやプログラムの構造が仕様を満たしているか等のプログラム記述について評価し、記録する。

(4) 評価に間違いや抜けがないかを再確認する。

(5) 全体の結果を元に配点を決め、学生毎に集計を行う。

2.3. 採点作業における問題点

(a) ソースの表示やコマンドプロンプトで対象フォルダに移動する作業やコマンド入力の作業は直接採点には関係なく単純な作業であるが、学生、設問を変える毎に行うので手間がかかり採点者に負担を与える。

(b) 採点者は通常、複数のウィンドウを開いた状態で採点を行う。コンパイルやテストプログラムを行い記録したり、ソースの内容を評価し記録したりと複数のウィンドウを切り替えて採点を行う必要がある。記録する場合には学生名簿から現在採点中の学生を探し入力を行うが、ウィンドウの切り替えが多いため異なる採点項目や学生に入力を行わないように毎回確認する必要が生じ作業効率を下げる。

(c) 特定の条件に一致するレポートのみを対象に採点や評価確認を行いたい場合、その条件に一致するものだけを抽出し作業することで効率が上がる。しかし条件に一致するレポートだけを抽出するのは非常に手間がかかり非効率なので、抽出作業を諦めざるを得ない。仮にグループ化を行ったとしても、レポートの評価が変わったときにグループの条件に一致したメンバーを正しく維持することが難しい。

(d) 採点終了後、評価項目に点数をつける必要があるので、各項目に当てはまる生徒の人数分布を調べ、配点基準を決める。分布を知るためには表計算ソフト等を用いて人数を確認しなければならず、手間がかかる。

(e) コンパイルを実行し、コンパイル結果の評価を行う。コンパイル結果は正常終了、ワーニング、エラーの 3 種類であり、これらを機械的に振り分ける必要がある。全学生、全設問に対してこの作業を行うので負担がかかる。

(f) プログラムを実行し、動作を確認する。プログラムの実行方法や動作は学生によって大きく変わる。例えば入力値の設定に関してもハードコーディングされているもの、引数で与えるもの、標準入力からの入力を行うものと様々であるので、プログラムの実行は機械的に行えない。

(g) 評価作業で最も労力を要するのがプログラムの内容を理解しながら評価を行う作業である。この作業は与える問題や言語によって評価する項目が変化する。

これらの問題点は採点を行う環境が整っていないために発生するものと評価を行う上で発生するものに分けることができる。(a)～(d)は採点環境の不備が問題であり、(e)～(g)は評価を行う上で発生する問題である。これらの問題を解決するために採点支援ツールを開発し、採点を効率よく行う環境の提供と評価作業の負担軽減のための支援を行う。本ツールは Java 言語で実装しており、採点データは C.S.E からダウンロードする“問題フォルダ¥設問フォルダ¥学生データ”のようなファイル構成のデータを対象としている。

3. 採点環境の提供

3.1. 採点作業負荷の軽減

採点作業では通常の統合開発環境とは違い、提出された 1 つ以上のソースファイルを設問や学生毎に区別して開く必要がある。本ツールは図 1 左側のようにファイル構成を崩さずにツリー構造で表示しているので、Java 言語のパッケージのようにファイル構成を意識する場合でも有効である。入力値や出力結果などが記述されているデータファイルを表示したい場合には拡張子を指定することでソースファイル以外のファイルも表示することができる。▲ボタンを押すことで前後の設問、学生へ順次移動でき、移動後ソースが自動で表示されるので、採点者は学生が変わる度にソースファイルを開く作業やフォルダを辿ってファイル構成を調べる作業を行う必要がない。移動後はコンパイルや実行コマンドが自動で入力される。このような機能により問題点(a)を解決する。

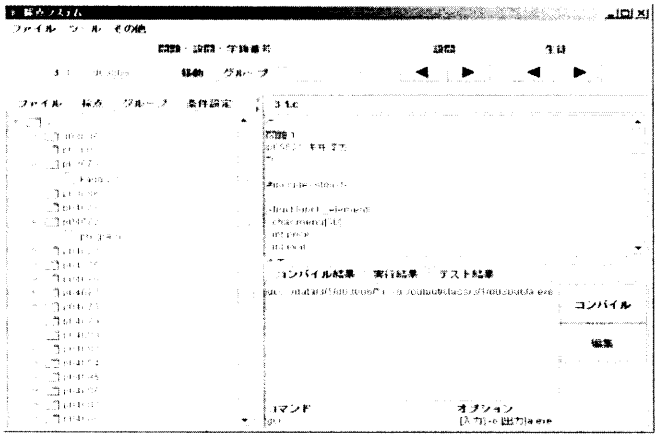

図 1　ファイル構造とソース表示画面

3.2. 画面切り替え作業の除去

本ツールは採点項目を決め、当てはまる採点項目をチェックすることで採点を行う。図 2 左側のように採点項目はグループ化されており「コンパイル結果」や「冗長性の有無」等の採点項目グループ中の当てはまる採点項目に対しチェックすることができる。採点項目グループ名や採点項目名は任意に決定でき、いつでも追加削除が可能である。採点項目グループ内で重複チェックを許す場合にはチェックボックスを、単一チェックしか許さない場合はラジオボタンが選択できる。採点項目は設問毎に独立しており、同一の設問であれば全生徒共通である。各採点項目には点数を付けることができる。

特定の学生のみを対象にした限定的な評価を行う場合は学生、設問毎にコメントとして記録が可能である。それ以外に学生にフィードバックするためのコメントも作成できる。コメントは学生、設問毎に点数をつけることできる。

またツール上からプログラムの実行ができる。図 2 右下はプログラム実行中の画面でインタラクティブなプログラムの実行にも対応しており、実行プログラムへの入力も行える。

このようにソースの表示、評価、コンパイルやプログラム実行も同一のウィンドウで行うことにより、採点中の学生の評価項目のみが表示されており、学生を間違えるようなミスを避けることができ、問題点(b)が解決できる。

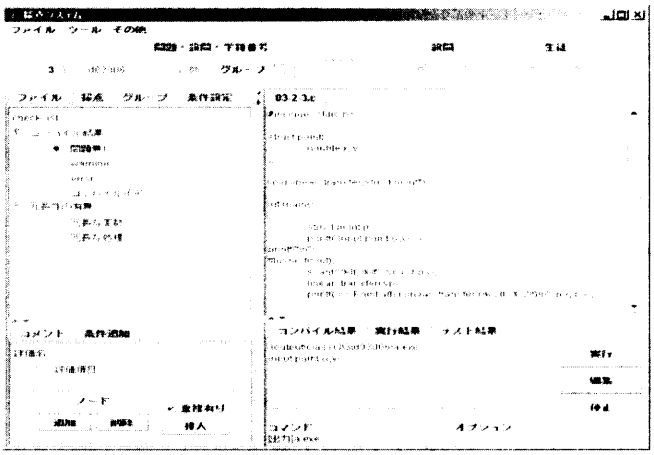

図 2　採点画面

3.3. 評価による絞込みと集計

本ツールでは評価項目を用いたグループ分けができる。図 3 左側はグループ分けの画面で、「コンパイル結果」内の「問題なし」にチェックが入っている生徒を「コンパイル可」グループとして抽出し、予めコンパイルが通らない生徒は評価しないようにしている。グループは複数の条件により作成することもでき、作成したグループのメンバーは動的に更新される。このように条件指定することで学生の絞り込みができる。グループ分けの際に属している人数をカウントすることも可能である。これにより問題点(c)、(d)を解決している。

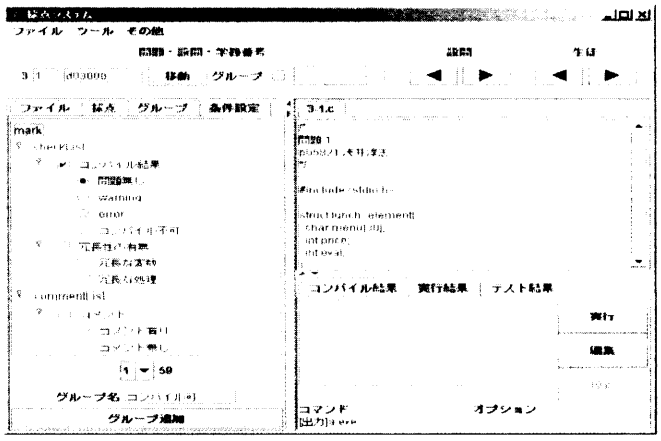

図 3　グループ条件設定画面

4. 自動化コンポーネントの組み込み

本ツールは評価作業を支援するためにツール上から外部プログラムを実行できる環境を提供している。実行環境は２つ用意されており、１つは外部プログラムを実行するだけの環境である。もう１つは外部プログラムを自動化コンポーネントとして実行する環境で、コンポーネントの出力結果を解析し、評価に反映することができる。後者の環境で外部プログラムを評価の自動振り分けを行う自動化コンポーネントとして組み込むためには、評価始める際に１つピリオド“.”を、続けて評価グループ名を[name(check or radio)：評価グループ名]のようなフォーマットで出力させ、本ツール上で実行すればよい。(check or radio)は評価項目が単一選択しか許さない場合に radio を、重複選択を許す場合には check を記述する。

4.1. 外部プログラムの実行

ツール上でプログラムの実行することで実行結果の評価ができる。実行コマンドは xml 形式の設定ファイルに拡張子とコマンド名の関連付けを行うことで設定ができる。学生によって実行方法や入力値の与え方が異なるので、実行コマンドはツール上で編集可能である。設問で実行方法を限定させない限り、問題点(f)については完全に解決することはできないと思われる。

4.2. コンパイラの実行

プログラムの文法に誤りがないか判断するにはコンパイルすればよい。コンパイラの出力は本ツールの自動化コンポーネントとして用いるためのフォーマットに変更することができない。しかし、拡張子が決まればコマンド名が決定できるだけでなくコンパイルコマンドが正常終了し、かつ出力がない場合には問題なし、正常終了し出力がある場合はワーニング、正常終了しなかった場合にはエラーのように評価方法が決定できる。そこで本ツールではコンパイラのみ外部プログラム実行環境とは別の環境で動作させ、コンパイラを自動化コンポーネントとして組み込んでいる。実行コマンド同様、コンパイラと拡張子の関連付けは設定ファイルで行う。コンパイル操作はソースが始めて表示される際に自動で実行と結果の記録が行われる。これにより問題点(e)のように採点者はコンパイル操作を意識せずに済み、チェックミスも防ぐことができる。また予め全ソースをコンパイルすることもできる。

4.3. xUnit の利用したコンポーネントの組み込み

Java 言語では JUnit[3]と呼ばれるテスティングフレームワークがある。この JUnit をベースに様々な言語に対応した xUnit が作成されている。例えば CppUnit や VBUnit[4,5]等が挙げられる。これらテスティングフレームワークはテストの前後処理や合否の判断方法を統一的に与えるもので、作成者は関数を実行するために必要な前後処理を記述した後、関数を呼び出し、その結果と期待値を比較することでテストを作成することができる。このときテストメソッドが呼び出す関数のシグネチャは学生全体で統一されている必要があり、問題を与える際に厳密に宣言しなければならない。更にテストの連続実行やテストが不成功の場合の情報提示機能も提供する。xUnit はテストが１つ実行される度にピリオドを１つ表示し、テストが正常に終了した場合はピリオド以外の出力はなく、実行結果が予期したものと一致しない場合は F（失敗）を表示する。プログラムが不正な処理を行い、例外を出力して終了した場合には E（エラー）を表示する。出力がピリオドのみの場合には問題なし、F が出力された場合には失敗、E が出力された場合にはエラーのように結果を振り分ける。

図４は JUnit を用いて２つのテストケースを実行した結果の一部である。１つ目のテストは成功しているが、２つ目のテストはテスト失敗の F が表示されており、テストが失敗したことを示している。

```
>java junit.textui.TestRunner AClassTest
. . F
Time: 0.015
There was 1 failure:
1)  testGetAdd(AClassTest    (略…
```

図４ JUnit 実行結果

図４から分かるように JUnit の出力は無駄なものを一切出力しない。JUnit は成功の場合、テスト回数分のピリオド、失敗やエラーの場合にはそれに加え、"E", "F"の表示とエラー箇所に対する情報だけを出力する。本ツールに適用する場合、この出力だけでは評価項目グループの検索、評価の振り分けができないのでテストプログラムに評価項目グループ名を出力させるように変更する必要が生じる。しかし本ツールで用いるためだけにプログラムを変更することは望ましくない。そこでソース変更を回避するために xUnit にアスペクト指向言語を用いて処理の追加を行う。アスペクト指向とはモジュール間に現れる横断的関心事をアスペクトとしてモジュールから分離するという考え方である。アスペクト指向はソースを変更せずにある動作(point cut)に対し処理の追加(weaving)ができる。これを使うことでテストプログラムを変更するこ

となく、新たな出力が追加できる。アスペクト指向言語として aspectj、aspectC++[6,7]等がある。これらを用いることでテストプログラムを本ツールから独立させることができる。

aspectj と JUnit を用いて出力結果を変更する例を示す。JUnit は TestRunner がテストプログラムを実行し、テストを行う。TestRunner は引数で指定されたクラスを辿り、名前が test で始まるメソッドを検索し実行するので test で始まるメソッドが実行される場所を point cut すればテスト結果に新たな出力を追加できる。また、aspectJ は対象プログラムのソースと共に ajc コンパイラでコンパイルすることで weaving が行える。図5は test で始まるメソッドを実行する際に実行したメソッド名を出力するためのアスペクトである。このアスペクトは任意の JUnit のテストプログラムに weaving することができ、一度図 5 のアスペクトを作成すれば、テストプログラムを入れ替え、ajc を用いてコンパイルを行うだけで aspectJ と JUnit を用いた自動化コンポーネントを作成できる。

```
public aspect TestAspect{
   before() : execution(void test*()){
      System.out.print("[name(check):"+
                              thisJoinPoint.getSignature()+"]");
   }
}
```

図5 aspectJ の例

図6は aspectj を組み込んだ際のテストプログラム実行結果で、実行されたテストメソッド名が出力されている。本ツールは採点グループ名 testGetDouble が成功、採点グループ名 testGetAdd はテスト失敗と読み取り自動振り分けを行う。

```
>java junit.textui.TestRunner AClassTest
. [name(check):  void AClassTest.testGetDouble()]. [name
(check):  void AClassTest.testGetAdd()]F
Time: 0There was 1 failure:
1)  testGetAdd(AClassTest)  (略…
```

図6 JUnit と aspectJ を用いた実行結果

このようにアスペクトを利用することで JUnit を用いた自動化コンポーネントを作成することができる。

5. 評価と考察

2つの授業で本ツールを用いて、実際にプログラムのレポート採点を行った。C 言語を対象にし、テストプログラムを用いずに採点を行った。採点終了後、4 人のツール利用者に意見を伺った結果、ツールのインターフェースが理解し易く、かつソースの表示やコンパイル作業が自動で行われ、作業が全て同一ウィンドウ上で行えるので作業の負担が軽減されたとの意見があった。一方で、ツールの操作性に問題があり評価項目の追加が面倒であるとの意見もあったので、操作については改善する必要がある。またツールの操作量を減らすにはテストプログラムを用いた採点の自動化が有効である。テストプログラムを用いた採点の自動化を行うことで採点者が行うべき作業が減り、更に作業負荷が軽減できると考えられる。また、本ツールの仕様ではプログラムの実行が確認できない範囲の問題があった。クライアント、サーバー間で通信を行うようなプログラムの場合、2つのプログラムを同時に実行しなければならず、本ツール上から実行することができない。このような場合には本ツールの機能の1つが使えない分、利便性が損なわれる。これを解決するには複数のコマンドを実行する環境を提供する必要があると考えている。

その他に採点項目の決定が難しいとの意見が多かった。評価項目を決定するには、採点者の経験を用いて決定する、要求仕様から満たすべき項目を挙げる、始めに 10 人程度のプログラムを大まかに見て、採点項目を決定するといった方法を用いることが多い。しかし、採点者が始めて扱う問題の場合や要求仕様で定めていない部分の評価には前述の決定方法は通用しない。この場合には、予め評価項目を決めるのではなくコメントとして残し、重複したコメントの学生が増えてきた場合にコメントが評価項目として設定するような機能があれば評価項目決定の支援が行えると考えている。

また、類似(盗用)レポート発見機能が欲しいとの意見も挙がった。厳密に類似レポートを発見するには、言語に依存した解析を行う必要がある[8]。本ツールでは言語に依存した解析を行う場合、外部プログラムを作成しなければならない。複数の条件を組み合わせたグループ化の機能を応用して、簡易的であるが類似レポート発見を行うことができる。これは学生がプログラムを盗用する場合の多くは変数名やコメント、改行の追加等により一見異なるソースに見せかけるだけであるからである。しかし、実行結果が適切であるか、仕様を満たしているかといった特徴を考えた場合には上記の偽装には影響されない。そこで、これらの条件でグループ化を行うことにより、盗用レポートである可能性の高いレポートを絞り込むことが可能になる。

レポートの主な評価項目はプログラムが実行でき、かつ正しく動作するか、仕様通りの記述であるかが挙げられる。プログラムが実行できるかを評価するにはコンパイルが有効である。本ツールは外部プログラムをコンポーネントとして実

行することで自動評価を行っている。この機能を利用し、コンパイルの自動評価を可能にした。また、正しく動作するかを評価するには実行結果を確認するのが良い。出題する設問にはテストケースの例を与えていることが多く、学生はその例を用いて実行ができるかを確認している。動作確認を行う際にはそのテストケースと出力結果を用いるとよい。なお、採点を行うにはその他のテストケースについてもテストを行う必要がある。全学生が統一されたシグネチャで実装することでテストプログラムが作成できるが、この場合は出題時に関数名、引数の型、戻り値、結果を全て指定する必要がある。テストプログラムを作成する場合にはテスティングフレームワークを使うことで、テストプログラムの作成、実行をサポートすることができる。JUnit で作成したプログラムと本ツールを独立させるには aspectj を用いる必要がある。xUnit とアスペクトを併用することにより本ツールに依存しないテストプログラムの利用ができる。仕様通りの記述であるかの評価についてはプログラムの内容を理解する必要がある。プログラムの特徴の解析方法を客観的に定義できる場合は、定義に基づいて解析プログラムを作成し、実行することで機械的に評価が可能である。しかし多くの場合、問題に依存した解析プログラムを作成するには多大なコストがかかる。

テストプログラムや解析プログラムは出題する問題や作成するプログラムに大きく依存するので、全ての学生に共通のコンポーネントを使うことは現時点では難しいと考えられる。しかし、問題の与え方を考慮し解析プログラム作成のためのフレームワークを提供することで自動化コンポーネントを利用できる環境を構築できると思われる。

6. 関連研究

6.1. 採点支援システム

採点システムの研究[9,10]では採点対象プログラムを解析、実行しその結果を元に自動評価を行っている。これらの研究の採点支援ツールは CASL や Java 言語と対象言語が固定されている。本研究では自動評価を行うツールを作るのではなく、採点作業を支援する環境の提供を行っている。また、自動化コンポーネントとして組み込むことで、自動採点が行えるようにした。つまり、本ツールは各言語に依存したツールと同様の解析、評価を行うことも可能である。自動採点を行う場合は採点方法が定義できなければならない。しかし実際に全ての項目を定義することは不可能であるので本ツールのように人手で採点を行い易い環境を提供することが必要である。

7. 今後の展望

本ツールはまだ C 言語のみでしか採点を行っていないので、今後 Java 言語でも本ツールを用いた採点を行い、言語による採点方法の違いやツールの有効性について調べる予定である。現時点では自動化コンポーネントとしてのテストプログラムがまだ作成できていないので、採点を自動化するためにどのようなコンポーネントが作成でき、それにより採点の作業負荷がどれだけ軽減できるかを調べる必要がある。

本ツールは C.S.E との Web サービスによる連携[2]を考えており、採点後には採点結果を C.S.E に送信し集計を行う予定である。それ以外にも採点中に付けた学生向けのコメントを C.S.E を通して学生にフィードバックしたいと考えている。

8. まとめ

本ツールを用いることでファイル表示やコマンド実行の作業を 1 つのウィンドウで行え、採点作業の負荷を軽減する環境を与えることができる。またコンポーネントを用いた自動評価ができる環境の提供を実現した。今後、コンポーネントとしてどのようなものが有効であるかを検討する必要がある。

C.S.E と連携によって採点結果の集計や学生側へのコメントのフィードバックを容易に行えるようになり、学習支援へと繋げることができるようになると考えている。

文献

[1]熱田智士, 松浦佐江子, "Java プログラミング演習向け課題レポート提出・管理機能を付加した授業支援システム", FIT2004 情報科学技術レターズ,Vol.3, pp.359-362, 2004

[2]熱田智士, 松浦佐江子, "授業の多様性に柔軟に適応する LMS の開発", 教育工学研究会, 2005

[3]JUnit: : http://junit.org/index.htm

[4]CppUnit : http://sourceforge.net/projects/cppunit/

[5]VBUnit : http://www.vbunit.com/

[6]aspectj : http://eclipse.org/aspectj/

[7]aspectC++ : http://www.aspectc.org/

[8]鷹岡良治, 脇田健, "静的解析のプログラム盗用発見への応用", 日本ソフトウェア科学会第 21 回大会, 7C-3, 2004

[9]内藤広志, "プログラミング演習の総合支援システムの概要", FIT2004", N-011, 2004

[10]渡辺博芳, 武井恵雄, "コース管理システム WebCT とプログラミング評価支援システムの連携", 情報処理学会第 67 回全国大会, 4D-7, 2005