# ソフトウェア開発技術者教育支援システム ALECSS のための プログラム点検用スクリプトの実装と評価

大月美佳[†1]　大田和樹[†2]　高崎光浩[†3]　掛下哲郎[†1]

**概要**：我々は各種の DevOps ツールを活用したソフトウェア開発技術者教育支援システム ALECSS を開発している．本システムは学生が提出したプログラムを様々な観点から自動的にチェックし，学生にフィードバックを返すことで，高品質ソフトウェアの開発技術の習得を支援すると同時に，教員による評価作業を支援する．ALECSS は，コンパイルチェック，実行結果のチェック，コーディングスタイルのチェック，コードの静的解析チェック等を行う．これらの機能を実現するために，継続的インテグレーションツール Jenkins とバージョン管理ツール Git を中心とし，様々な DevOps ツールを活用して本システムを構築する．コンパイルチェックやコーディングスタイルチェックなどの評価は既存ツールでおこなえるが，実行結果のチェックや Git の作業状況チェックなどの評価は独自のスクリプトを用意する必要がある．これらスクリプトでは，チェック内容が演習内容や学生の情報に依存するため，比較のための模範解答をテンプレートから生成しなければならない場合もある．本稿では，独自スクリプトを必要とするチェックのうち，ファイル構成チェックおよび出力結果チェックについて，スクリプトの実装とその評価について述べる．実装にあたっては，演習内容を柔軟に書けるように，単なる文字列の置き換えだけでなく前処理を書けるようなテンプレート形式を採用した．評価にあたっては，昨年度の演習結果を使った試行をおこない，出力結果の揺らぎによる誤判定がどの程度見られるかを確認し，対応策を考えた．

**キーワード**：DepOps ツール，教育支援システム，ソフトウェア品質，協同開発

# Implementation and Evaluation of Program Checking Scripts for Software Engineer Education Support System ALECSS

MIKA OHTSUKI[†1]　KAZUKI OTA[†2]　MITSUHIRO TAKASAKI[†3]
TETSURO KAKESHITA[†1]

***Abstract***: We are developing an education support system to train software developers utilizing several DevOps tools "ALECSS." The system automatically checks programs submitted by students and return their feedbacks to them to support learning development techniques for high quality software, as well as to support evaluation by the teacher. ALECSS executes compile checking, execution result checking, coding style checking, static code analysis etc. In order to realize these functions, we build up the system mainly by utilizing DevOps tools such as continuous integration tool Jenkins and version control tool Git. Existing tools can be applied for evaluation such as compile checking and coding style checking. However, other evaluation such as output checking and Git working status checking require scripts developed uniquely. Since these scripts uses exercise contents and student's information in checking, sometimes need to generate typical results from templates for comparing them with the students' answers. In this paper, we propose and evaluate scripts for file structure checking and output checking. On implementing the scripts, we adopt template format which can not only replace strings simply but also write preprocessing strings in order to prepare exercises flexibly. On evaluating the scripts, we execute them using last year data on trial, then we check how many errors based on fluctuation of output and consider measures for the errors.

***Keywords***: DevOps Tools, Education Support System, Software Quality, Cooperative Development

## 1. はじめに

近年，ソフトウェアの大規模化に伴い，複数人で開発を行う協同開発が一般的になっている．そのような協同開発においてチームは様々な問題に直面する．例えば，課題をメンバ間で適切に共有できず，進捗が見えにくくなることや開発内容が競合することなどである．そういった問題に対して，開発現場では，各種の DevOps ツール[1,2]を活用した自動化により解決を試みている．我々は，これらの DevOps ツールを教育目的で活用することを考えた．

佐賀大学理工学部知能情報システム学科では，ソフトウェア協同開発を体験学習するために「システム開発実験」が開講されている．本科目は，3 年生を対象とする全 15 回の演習科目である．本科目ではこれまで，学生が提出したコードを教員が手作業でチェックしていた．そのため，教員側にも大きな負担がかかり，学生にフィードバックを返すまで時間がかかっている．同様の課題は，プログラミング教育等で多数見られる．

そこで，我々は，DevOps ツールを用いて学生が提出したコードを様々な観点から自動的にチェックし，学生にフィ

†1 佐賀大学大学院工学系研究科
Graduate School of Science and Engineering, Saga University.
†2 佐賀大学大学院医学系研究科
Graduate School of Medical Science, Saga University
†3 佐賀大学医学部附属病院
Saga Medical School Hospital, Saga University

ードバックを返すソフトウェア開発技術者教育支援システム ALECSS（Automated Learning and Evaluation Cycle Support System）を提案している[3]．ALECSS を活用することで，学生（個人およびチーム）は作成したコードを迅速にチェックし，直ちに改善作業を行える．一方，教員も定型的な確認の手間を削減すると同時に，学生やチームの進捗状況を容易に確認できる．

我々はこれまで，[3]において，システム開発実験の新たな授業設計と，ALECSS の構想およびシステムを構成する各種の DevOps ツールの設定を提案した．本論文では，この構想に基づきつつ，設計の詳細化とこれまでの授業で収集したデータを活用した予備実装，特に独自開発したチェックツール部分について説明する．

本論文の 2 節では，これまでのシステム開発実験の概要と ALECSS で対象とする学生が提出したプログラムに対する評価項目について説明する．3 節では，2 節で挙げた評価項目について，プログラミングスタイルチェッカー Checkstyle [7]，プログラム静的解析ツール FindBugs [8]，ユニットテストツール JUnit [9]，および独自に開発するシェルスクリプトと，それらを駆動する Ant の設定ファイルの設計について説明する．4 節では，独自に用意する必要があるスクリプトのうち，ファイル構成チェックと出力結果チェック用のものについての設計と実装について述べる．5 節で，前年度の提出プログラムに対してこの一部実装したスクリプトを試用してみた結果について紹介し，評価をおこなう．6 節はまとめである．

## 2. システム開発実験と評価項目

### 2.1 システム開発実験

佐賀大学理工学部知能情報システム学科では，平成 17 年度から，ソフトウェア協同開発を体験学習するための必修科目として 3 年生を対象として「システム開発実験」という演習科目を開講している．

本実験では実際の開発現場で使用されているツールを用いて，小規模のチームで開発を行うことで，企業で行われているソフトウェア協同開発を疑似体験することを目的としている．開発するソフトウェアは簡単な画面遷移プログラムであり，1 名のリーダと 1 名の副リーダ（トラッカー）を中心とする 8 名程度のチーム開発体制を取る．学生の開発環境は Windows OS を前提とし，バージョン管理ツール Git，統合開発環境 Eclipse，単体テストフレームワーク JUnit および進捗管理にツリー型掲示板を使用している．

全 15 回の授業のうち，ツールの利用方法を中心とする個人演習が 5 回，グループ演習が 10 回となっている．グループ演習はそれぞれ 4 回から成る 2 つのイテレーションに分かれており，1 つのイテレーションの実装期間は 3 週間（3 時間×3 回）で，残りの 1 回は報告と相互評価を行う回である．開発グループには，各回の終わりに push（遠隔リポジトリへの提出）と作業報告（ツリー型掲示板への記事投下）が義務づけられている．実装にあたっては，先にテストを書いてから実装をおこなうテスト駆動開発[9]および，1 台の PC を 2 人（ドライバとナビゲータ）で使用するペアプログラミング[10]が推奨されている．演習の構成を図 1 に示す．

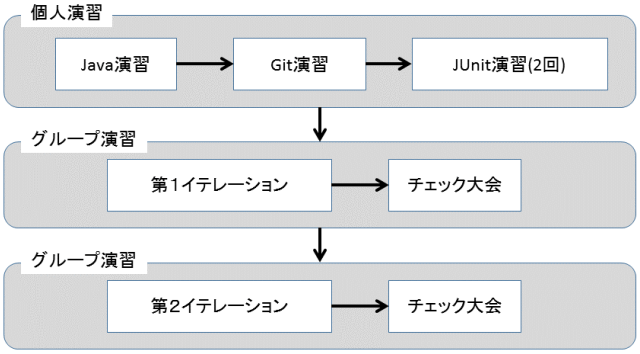


図 1　現在の演習の構成

Figure 1　Current Structure of Exercises.

個人演習では，インストール回の後， Java 演習 1 回，Git 演習 1 回，JUnit 演習 2 回を行う．システム開発実験の各演習では，表 1 に示す各種の評価項目を設定している．なお，今期平成 28 年度までは Java 演習を Git 演習に先行しておこなっているが，自動化を図るにあたって，今後 Git 演習を先行させる予定である，

グループ演習では，毎回の報告を確認しつつ，イテレーションの最後にプロジェクト全体について上述の評価を行う．また，グループ内での報告をさせ，それをメンバ間で相互評価する．

### 2.2 提出プログラムの評価項目

システム開発実験の各演習では，表 1 に示す各種の評価項目を設定している．各演習には重複した部分があるため，いくつかの評価項目は別の演習でも評価されうる．その対応を図 2 に示す．

表 1　チェック名と評価項目

Table 1　Check Name and Evaluation Items

| チェック名 | 評価項目 |
|---|---|
| ファイル構成チェック | ファイルが全て提出されているか |
| コーディング標準チェック | コーディング標準に準拠しているか |
| コンパイル可能チェック | プログラムがコンパイルできるか |
| 出力結果チェック | プログラムの出力結果は正しいか |
| Git 作業実行チェック | 指定した作業を全て行っているか |
| テスト実行チェック | 全てのテストが成功するか |
| テストケース空実装チェック | テストケースが空でないか |
| テストコード有効性チェック | テストコードが何でも通してしまわないかどうか |
| バグチェック | 典型的な落とし穴に陥っていないか |

Java 演習では，ファイル構成チェックで各課題に必須と指定されたファイルがあるかどうかを確認後，コーディング標準チェックで準拠しているかを評価し，コンパイル可

能かチェックする．この 3 つのチェックは，他の演習でもおこなう．出力結果が望ましいかは，Java 演習でしか求めていないので，この演習だけでチェックし，個人演習期間中は提出遅れに対応するだけである．Git 演習での課題作業確認も，この演習でしか行わないため，以降の演習では再提出確認のみである．ここでは再提出確認のみは破線の○で示している．JUnit 演習では，ファイル構成チェック，コーディング標準チェック，コンパイル可能チェックに加えて，JUnit のテスト実行，テストケースの中身が空でないか，テストコードが有効かどうかをチェックする．ここで「有効かどうか」というのは，「失敗すべき時に成功してしまわないか」ということであり，テスト設計として有効かどうかということではない．

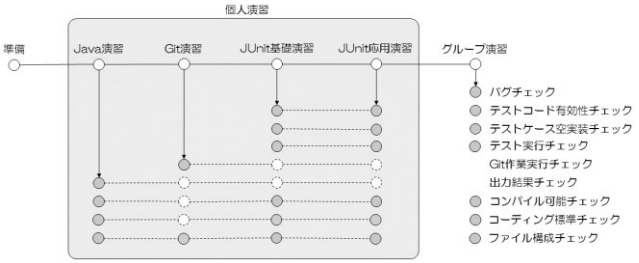


図 2　演習とチェックの対応

Figure 2　Correspondence between Exercises and Checks

グループ演習では，標準出力への文字出力はおこなわないため出力結果チェックは不要である．また，Git 演習のみの課題である Git 実行チェックもおこなわない，一方，複数のファイルからなるプログラム全体での「ありがちな間違い」を検出するために，コード静的解析ツールを利用したバグチェックをおこなう．

ALECSS では，これらのチェックをできる限り自動化することを考える．

## 3. ALECSS での評価自動化

[3]で Jenkins などの DevOps ツールを利用して，2.2 節で述べた評価項目の自動化を検討した．図 3 に ALECSS の全体構想を示す．統合開発環境の Eclipse から学生が Git リポジトリに提出したプログラムについて，その提出をトリガーに，または提出期限に基づいて，Jenkins から Ant を起動し，ソフトウェアテスト環境の JUnit，コーディングスタイルチェッカーCheckstyle, 静的コード解析ツールの Findbugs などを駆動することを考えている．ここでは，[3]での Ant の設定ファイル build.xml を用いた各チェック自動化の検討内容について簡単に述べるとともに，検討を進めて追加変更した部分，特に教育用に開発した独自チェックツール群（右下灰色部）について述べる．

### (1) ファイル構成チェック

Ant のタスクとして，condition タグおよび available タグを利用することで固定名称のファイルの存在チェックができる．しかし，ファイル構成チェックはすべての演習で必要とされており，演習ごとにその構成が異なる．また，ファイルやフォルダの中には学生の学籍番号やプロジェクト名に依存して名称が変わるものもある．このため，演習・学生・プロジェクトごとに対応する必要がある．このため，これらに対応できる build.xml およびスクリプトを演習・学生・プロジェクトごとに初期生成する．

### (2) コーディング標準チェック

taskdef で定義することで，Ant のタスクとして，コーディングスタイルチェッカーCheckstyle を起動できることを確認しており，講義で使用しているスタイルを標準で配布されているもの（Sun Code Conventions または Google Java style）に準拠させればそのまま使用できる．

### (3) コンパイル可能チェック

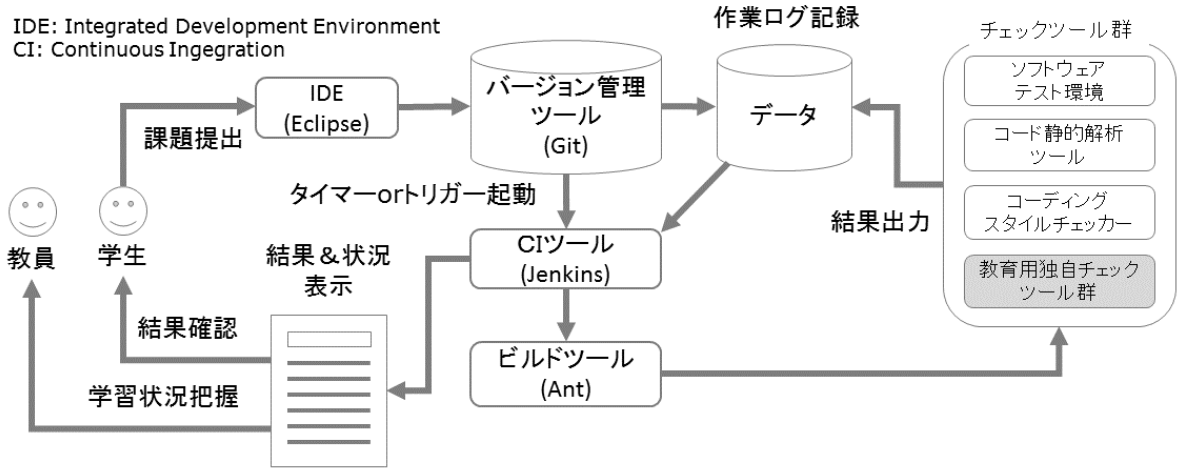


図 3　ALECSS の全体構想

Figure 3　Entire Structure of ALECSS

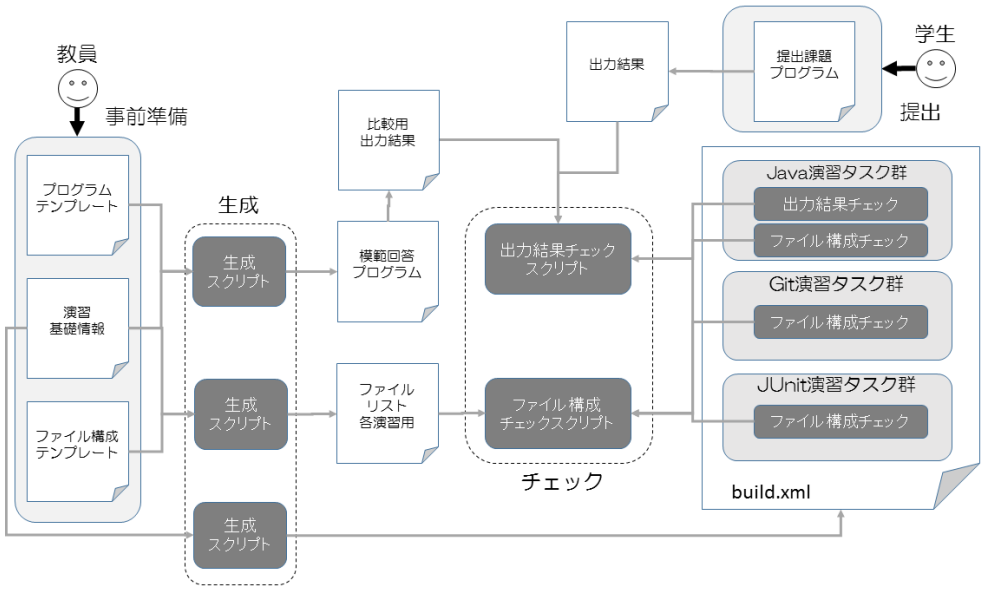


図 4　出力結果とファイルリストの生成

Figure 4　Generation of Output Result and File Lists.

Ant のタスクで，javac タグを使用してコンパイルを実行でき，その結果を Jenkins 上で表示できる．

(4)　**出力結果チェック**

java タグを利用すれば，Java プログラムは実行でき，そのログを記録し，Jenkins 上で表示することもできる．さらに自動化を進めるために，出力結果が予め用意したものと同じかどうか比較するスクリプトについて検討した．結果が固定のものであれば diff を使用するのが簡単であるが，実際の演習では，安易なコピーアンドペーストの防止のため，学籍番号や氏名に依存した結果を出力する課題もあり，このような課題については，(1)の場合と同様に，比較用の結果を初期生成しておく必要がある．

(5)　**Git 作業実行チェック**

git log コマンドにてコミットログを取得し，課題作業として指定されている「ファイル追加(Add してコミット)・削除(Remove してコミット)・更新（編集してコミット）・前コミットへの復帰(Revert)」の 4 つがおこなわれていることを確認する．Ant 自体にはこのような機能がなく，また，Git のコミットログにもコマンド自体は記録されないため，これら課題作業がおこなわれているかチェックするスクリプトを独自に作成し，それを呼び出して実行するタスクを定義する．

(6)　**テスト実行チェック**

JUnit は junit タグを使用してタスクとして設定することができるのでこれを利用して，ログ出力をおこない，Jenkins 上で成功数を確認する．

(7)　**テストケース空実装チェック**

JUnit フレームワークでは，テストケースにコードを何も書かなくても成功にできてしまうため，そのような実装を発見するために，各テストケースメソッドの実装行数を確認する必要がある．そのためにテストケースメソッドを抽出する簡易の Java 言語パーサを用意する．そして，そのパーサで抽出した行数をカウントするような独自スクリプトを用意し，タスクとして定義する．

(8)　**テストコード有効性チェック**

上述したように，ここでの有効性チェックというのは，どんな実装に対してもテストが成功してしまうようなテストケースになっていないかをチェックするものである．テストの意味解析をおこなうのは困難であるため，テストが失敗するようなテストコードを別途用意しておき，提出されたテストケースのファイル群を失敗用テストコードの置かれた作業領域へコピーしてコンパイル・実行し，それが失敗するかどうかを確認する．

(9)　**バグチェック**

個人演習でもおこなう一連のチェックに加えて，グループ演習では全体的に「よくありがちな間違い」をチェックする．このために，静的解析ツールの Findbugs を導入し，Ant のタスク定義をおこなって実行する．

各チェックとその自動化を検討した結果，(2)コーディング標準チェック，(3)コンパイル可能チェック，(6)テスト実行チェック，および(9)バグチェックはほぼ既存ツールだけの利用で可能だが，(1)ファイル構成チェック，(4)出力結果チェック，(5)Git 作業実行チェック，(7)テストケース空実装チェック，および(8)テストコード有効性チェックについては独自のスクリプト開発が必要だということがわかった．

このうち，(1)のファイル構成チェックは，演習で構成が変わることや，ファイルやフォルダ名が学籍番号や氏名，

プロジェクト名に応じて変わることがある．また(4)の出力結果チェックにも学籍番号や氏名に依存するものがある．これらに対応するには，ファイルリストファイル，出力結果ファイル，Ant 設定ファイルの生成が必要である．この 2 つのチェックについては，次節にて詳細を述べる．

(5)の Git 実行チェックおよび(7)(8)のテスト関係チェックについては，現在検討中である．Git 実行チェックでは，git log で収集できる作業ログから課題作業に該当する文字列の抽出をおこなってそれをチェックする予定である．テストケース空実装チェックでは，簡易パーサでテストケースコード部を抽出し，空でないか判定する予定である．テストコード有効性チェックでは，失敗するテストケースコードの事前用意や提出されたファイル群からのコピーなどの処理をおこなう予定である．これらについては稿を改めて紹介したい．

## 4. 生成およびチェックスクリプト

本科目は従来，学習支援システム Moodle で資料配布や課題管理をおこなってきた．学生は本校の統合認証システムを経由して講義ページに登録し，資料や課題にアクセスする．ALECSS は現在 Jenkins を中心に構成しているが，将来的には Moodle 上の登録情報を利用し，Moodle のモジュールを介して ALECSS の設定ができたり，ALECSS でのチェック結果などを Moodle の講義ページからも確認できたりすることが望ましい．このため，スクリプトの実装を，Moodle の実装言語である PHP でおこなうことにした．

以下では，出力結果チェックとファイル構成チェックに必要な生成およびチェックのスクリプト，およびそれに関連して生成が必要となる build.xml の生成スクリプトの設計と実装について述べる．これらの概要を図 4 に示す．

講義担当教員が予め学籍番号や氏名など必要な情報を記載した演習基礎情報ファイルと，それらを置き換えるように記述したファイルリストテンプレートおよび出力結果テンプレートを用意することで，システムは各学生の提出物と比較するために必要な模範解答ファイル群を生成する．学生が課題を提出した後，締め切り時点でこれら模範解答と提出物をチェックするスクリプト各演習のタスクから呼び出して評価の基盤となる結果を生成する．

### 4.1 出力結果の生成

上述したように，出力結果チェックでの出力結果は，コピー対策のため，学生ごとに異なることがある．現在は，学籍番号や氏名(ローマ字)に依存した違いとなっている．例えば，学籍番号「16233001」，氏名「山田太郎（Taro YAMADA)」という学生の場合，これまでの Java 演習で期待される出力は，図 5 のようになる．Java 演習は，氏名表示課題，Stack 課題，javadoc 生成課題の 3 つの小課題から成っており，出力結果の比較は氏名表示課題と Stack 課題の 2 つについて必要である．

氏名出力課題に対応する一行目は固定の文字列である「Hello」にローマ字氏名がついただけであるが，Stack 課題に対応する 4 行目末尾の 1 という数値は，「学籍番号下 3 桁を 10 進数としたもの」という指定をしている．3 行目はそれに 10 を足したもので，2 行目は 20 を足したものである．このようにプログラムによっては，内部で条件判定や計算がおこなわれた結果が出力されるため，単純な文字列や数値の置き換えでは済まないことがある．

```
Hello Taro YAMADA
name: No3, value: 21
name: No2, value: 11
name: No1, value: 1
```

図 5　出力結果例

Figure 5　A Sample of Output Result.

当初は図 5 のような出力結果のテキストをベースとするテンプレートを用意しておき，その中で文字列を置き換えることを考えたが，図 5 の 3 行目以降で必要となる計算式は別途定義する必要が生じる．演習のため，高度に複雑な計算式が必要となる可能性は低いが，条件式などが入ってくれば計算能力としては通常の言語に近くなってくる．テンプレートのためにそのような独自の定義用言語が必要となると，設定をおこなう教員の負荷が増大してしまう．別途模範解答としてプログラムを用意するのであれば，その出力を利用した方が，教員にとっては簡便である．

つまり，テンプレートは模範解答として用意するプログラムに変数依存部分だけを埋め込んだものとし，別途定義した変数である学籍番号や氏名などの情報を挿げ替えて生成したプログラムをコンパイル，出力し，その結果を期待される出力結果とした方が望ましい．

このため，プログラムテンプレート自体を PHP スクリプトとして記述する．つまり，プログラム本体はヒアドキュメント，置換が必要な変数はそこへの埋め込み変数とし，「学籍番号下 3 桁を 10 進数としたもの」のような処理が必要な場合は，ヒアドキュメントの前で独自の変数を定義し，PHP のコードを記述して処理するものとする．

表 2　テンプレートで使用可能な置換変数

Table 2　Replacement valuables for templates

| 置換変数 | 意味 |
|---|---|
| $EXID | 課題の ID。基本的には回数（例：1） |
| $EXDATE | 課題の出題日（例：2016/04/26） |
| $SID | 学生の ID。基本的には数値記号列（例：16233001） |
| $SNAME_JP_FIRST | 学生の日本語でのファーストネーム(例:太郎) |
| $SNAME_JP_LAST | 学生の日本語でのラストネーム（例：山田） |
| $SNAME_JP | 学生の日本語での氏名 $SNAME_JP_LAST $SNAME_JP_FIRST で定義される |

| | |
|---|---|
| $SNAME_EN_FIRST | 学生の英語でのファーストネーム（例：Taro） |
| $SNAME_EN_LAST | 学生の英語でのラストネーム（例：YAMADA） |
| $SNAME_EN | 学生の英語での氏名<br>$SNAME_EN_FIRST $SNAME_EN_LAST で定義される |

現在の置換変数としては，表 2 のものを使用することができる．この置換変数を利用して書いたプログラムテンプレートの一部を図 6 に示す．これを PHP で実行すると，図 7 のようなコードが生成される．

これを用いて，出力結果の生成を以下のようにおこなう．

```
<?php
// 出力指定
$dir = $DIST_DIR . "/MyTest$SID";
if (!file_exists ( $dir )) { mkdir($dir, 0755, true); }
$OUTFILE = $dir . "/MyTest$SID.java";

// 学籍番号の下3桁を十進数にする
$num = intval(substr($SID, 5, 3));

$data = <<<_ALX_JAVACODE
/*----------------------------------
 * 佐賀大学理工学部知能情報システム学科
 *   システム開発実験 第 $EXID 回 演習課題 その1
 * 作成者: $SID $SNAME_JP
 * 作成日: $EXDATE
 *----------------------------------*/
import mypack.*;

/**
 * mainメソッドを実行するクラス
 * @author $SNAME_JP
 */
public class MyTest$SID {
        /**
         * mainメソッド
         * @see MyObject
         * @see MyStack
         */
        public static void main(String[] args) {
                // 文字列Helloに氏名を加えたものの表示
                System.out.println("Hello $SNAME_EN");

                // 学籍番号の下3桁を十進数にしたものを変数numberに代入
                int number = $num;
```

図 6　プログラムテンプレート例

Figure 6　Sample of program template.

```
$ /c/dev/xampp/php/php.exe MyTest_SID_.php
/*----------------------------------
 * 佐賀大学理工学部知能情報システム学科
 *   システム開発実験 第 1 回 演習課題 その1
 * 作成者: 16233001 山田 太郎
 * 作成日: 2016/04/19
 *----------------------------------*/
import mypack.*;

/**
 * mainメソッドを実行するクラス
 * @author 山田 太郎
 */
public class MyTest16233001 {
        /**
         * mainメソッド
         * @see MyObject
         * @see MyStack
         */
        public static void main(String[] args) {
                // 文字列Helloに氏名を加えたものの表示
                System.out.println("Hello Taro 山田");

                // 学籍番号の下3桁を十進数にしたものを変数numberに代入
                int number = 1;
```

図 7　プログラム生成例

Figure 7　A Sample of Program Generation.

1. 指定された学生について設定情報から置換変数の値を取得する
2. 指定されたプログラムテンプレートを PHP のプログラムとして実行し，1 の情報で置換された Java プログラムを生成する
3. 2 で生成したプログラムをコンパイル・実行し，出力結果を学生ごとのファイルに保存する．

### 4.2 ファイルリストの生成

プログラムの生成と同様に，ファイル名やフォルダ名に学籍番号や演習番号などを指定することはあり得る．これまでの演習では，フォルダとメインのファイルに学籍番号を含むような指示を出していた。このため，リストを用意するにしても，4.1 と同様に置換が必要となる．

4.1 で生成するプログラムの構成から抽出することも考えたが，プログラム以外も含み得ることから，テキストとして入力することとした．置換変数としては表 2 のものが使用できる．図 8 に Java 演習でのファイル構成リストのテンプレートの一部を，図 9 に置換結果を示す．

置換結果を学生ごとのファイルに保存して，ファイル構成チェックに利用する．

```
MyTest$SID/src/MyTest$SID.java
MyTest$SID/src/mypack/MyObject.java
MyTest$SID/src/mypack/MyStack.java
(以下略)
```

図 8　ファイル構成リストのテンプレート（一部）

Figure 8　Part of template of file structure list.

```
MyTest16233001/src/MyTest16233001.java
MyTest16233001/src/mypack/MyObject.java
MyTest16233001/src/mypack/MyStack.java
(以下略)
```

図 9　置換結果

Figure 9　Result of replacement.

### 4.3 build.xml の生成

```
<?php
// 出力指定
$dir = $DIST_DIR;
if (!file_exists ( $dir )) { mkdir($dir, 0755, true); }
$OUTFILE = $dir . "/build.xml";

$stu_src = $STU_DIR . "/src";

$data = <<<_ALX_BUILD_XML
<?xml version="1.0" encoding="UTF-8" ?>
<project name="autocheck" default="allcheck">
  <target name="allcheck">
    <antcall target="check_filelist"/>
    <antcall target="compile_check"/>
    <antcall target="check_output"/>
  </target>
  <target name="check_filelist">
    <exec executable="$_ALX_HOME_/scripts/check_filelist.bat" dir="./" output="filelist_check.txt">
      <arg line="filelist.txt"/>
    </exec>
  </target>
```

図 10　build.xml のテンプレート（一部）

Figure 10　Part of template of build.xml.

出力結果のチェックおよびファイル構成チェックは，学生ごとにそれぞれ個別の Ant タスクとして実行される．4.2 で述べたように，フォルダ名やファイル名に学生ごとに異なるような情報が入る可能性があるため，build.xml も学生ごとに生成する．

出力結果チェックは，学生ごとに生成した模範の出力結果と，実際の出力結果を比較した結果をログとして保存する．また，ファイル構成チェックも，同様に，学生ごとに生成した模範のファイル構成リストと実際のファイル構成状況を比較した結果をログとして保存する．このようなタ

スクを学生ごとに実行できるように，図 10 のようなテンプレートとして記述しておく．

### 4.4 チェックスクリプト

出力結果チェックとファイル構成チェックは個別のスクリプトを用意しておこなう．当初は diff コマンドでの比較で十分かを検討したが，比較時の揺らぎへの対応や結果の集計処理をし，また将来的に Moodle などから利用可能とするために PHP でのスクリプトを用意することにした．

比較時の揺らぎというのは，例えば氏名のローマ字記述においての大文字小文字の揺らぎや単語間の空白の違いのことである．また，結果の集計ではそのような違いがどの程度解答状況に影響しているかを見るために文字数などをカウントできるようにしておきたい．

このため，出力結果チェックスクリプトにおいては，(1)大文字小文字の区別をする／しないを選択可能に，(2)空白を無視する／しないを選択可能に，(3)空白ありの文字数と空白抜きの文字数を比較結果に出力するようにした．

## 5. 試行と評価

出力結果チェックとファイル構成チェックをおこなう Java 演習について，昨年度の提出物を対象に 4 節で設計・実装した生成およびチェックスクリプトの試行をおこなった．試行においては，Jenkins への組み込みはおこなわず，試行用の駆動スクリプトで対象提出物すべてを処理して集計した．置換変数に指定する必要のある学生のデータは CSV ファイルとして用意し，初期化時に読み込んだ．

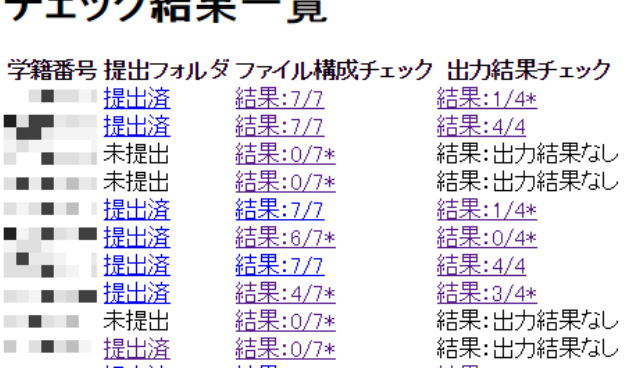


図 11 チェック結果一覧ページ

Figure 11 List page for check results.

暫定的に出力結果を一覧から閲覧できるように HTML ファイルを作成したが，実際の表示については，今後 Jenkins への組み込みをおこなって，学生へのフィードバックもできるようにしたい．今回暫定で作成した表示画面を図 11 に示す．一致行数が元の行数に満たないものについては「*」を付し，結果へのリンクをチェックしやすいようにしている．

### 5.1 チェックスクリプトの試行

昨年度の提出物について，出力結果チェックを適用した結果を表計算ソフトで集計したものを図 12 に示す．なお，前述したように氏名表示課題では 1 行，また Stack 課題では 3 行の出力がある．

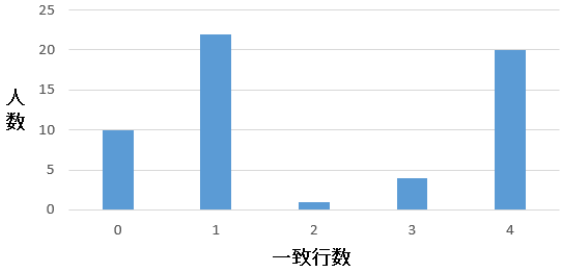


図 12 出力結果チェックの分布

Figure 12 Distribution of output check results.

未提出と構成ミスを除く提出 57 件のうち，4 行中 4 行すべて一致するものが 20 件，3 行一致が 4 件，2 行一致が 1 件，1 行一致が 22 件，すべて不一致が 13 件となった．これらの内訳を図 11 の各リンクから目視で確認すると，以下のようなことがわかった．

1. 1 行のみ一致では，氏名表示課題は正解しているが，Stack 課題で間違っている．間違いには，数値を間違えているケースと書式を間違えているケースの大きく 2 種類がある．数値を間違えている方が問題である．
2. 3 行一致では，Stack 課題は正解しているが，氏名表示課題で間違っている．間違いには，そもそも氏名ではない，氏名の一部しかない，日本語になっている，と大きく 3 種類がある．
3. 2 行一致は 2 の 3 件一致の亜種で，Stack 課題のうち一か所を間違えているケースである．

3 のケースの出力結果を例として図 13 に示す．見てのとおり，No3 となるべきところを No2 と入力ミスしている．

```
大文字小文字区別：なし
空白文字の違いを無視：あり

1: -4
Hello SHOUMA KAWANO
Hello World
2: 1
name: No3, value: 37
name: No2 , value: 37
3: 0
name: No2, value: 27
name: No2 , value: 27
4: 0
name: No1, value: 17
name: No1 , value: 17

RESULT:4,4,2,79,74,68,61
```

図 13 出力結果チェックの表示例

Figure 13 Example of output check.

ファイル構成チェックについても同様の分析をおこなった．ファイル構成チェックで確認するファイルは，氏名表示課題と Stack 課題の 2 つで実装しなくてはならない 3 つの java ファイル，それらをコンパイルした 3 つの class ファイル，および javadoc 生成課題で生成される index.html ファイルの 7 つである．7 つのファイルのうちすべて揃っているのが 51 件，6 つが揃っているのが 4 件，5 つと 4 つがそれぞれ 1 件だった．なお，まったく揃ってない 0 が 10 件あった，これらの内訳を図 11 のリンクから見てみると，

以下のようなことがわかった．

1. 0 のケースには未提出 5 件以外に，構成ミスが 5 件あった．Eclipse を使用しているならばおこりにくいミスのため使用していない可能性がある．
2. ほとんどの提出物のファイル構成は揃っており，6 つのものは javadoc 演習を実行していないものだった．
3. 4 つのケースは class ファイルなしで，5 つのケースはクラス名の綴りミスだった．

### 5.2 評価

5.1 の試行を通して，プログラム生成，ファイルリスト生成，build.xml 生成の 3 つの生成スクリプト，および，出力結果チェックおよびファイル構成チェックの 2 つのチェックスクリプトについて評価をおこなった．生成スクリプトについては，現況では Java 演習という 1 つの演習情報にしか対応していないので，Git 演習や JUnit 演習の追加情報にも対応できるように拡張していく必要がある．

チェックスクリプトに関しては，ファイル構成チェックについては特に問題はなかったが，出力結果チェックでいくつか問題が見られた．具体的には，氏名表示課題で求められていたローマ字記述に対して学生によって綴りに揺らぎが見られた．例えば「こうへい」という読みに対して「KOUHEI」と書くケースと「KOHEI」と書くケースなどがあった. この揺らぎにより不正解とされるケースが 22 件あったため, 今回は CSV データの修正をおこなって対応した．このような表記問題はローマ字に限らず，日本語でも起こりうるため，置換で使用する表記情報については予めシステムに登録しておく必要があるだろう．

その他，出力結果チェックでの書式の空白有り無しおよび大文字小文字の揺らぎについては，フラグで調整できるようにしており，今回は空白の差異および大文字小文字の違いも無視して評価した．しかし，これ以外に「:」(コロン)が「;」（セミコロン）になっていたり，文末に「.」（ピリオド）が付されていたりしたために不正解とされているケースが，それぞれ 1 件ずつ見られた．どの程度の揺らぎまで許容するかは採点者の判断に寄るため，システム的には一致度の表示をおこなう程度にとどめ，最終的な採点を支援する形にしておくのが望ましいと考える．

## 6. おわりに

Jenkins 上での統合を目指して，個人演習のうち Java 演習で使用する出力結果チェックおよびファイル構成チェックのための生成スクリプトとチェックスクリプトの設計と実装をおこなった．そして，昨年度の演習の提出物について，これら 2 つのチェックを試行し，その結果からスクリプトの評価をおこなった．出力結果については，学生の実装の揺らぎにより，不正解とするか判断のわかれる結果もあったが，採点作業で手間のかかる部分を自動化することにより教員の負荷はある程度削減できたと考える．

今後は Jenkins への組み込みにより更なる自動化を図るとともに，集計機能の充実による学習支援や学生へのフィードバック機能の実現を図っていきたい．また，今回実装した以外の Git 演習や JUnit 演習で必要な独自スクリプトについても設計と実装を進めていきたい．

出力結果チェックについては，プログラムの自動採点という部分だけを見ると, 既存研究として[11-13]などがある．今回開発したツールがそれらと違うのは，プログラムの自動採点だけではなく，講義で必要とされるチェック機能を Ant から呼び出せるようなスクリプトとして構築していることである．既存のコーディングチェックルーツと同様の方法で呼び出せるような設計にすることで，Java 演習における教員の定型チェック作業の一部である出力結果チェックおよびファイル構成チェックの自動化が ALECSS というプラットフォーム上で可能となった．

なお，現在の実装では機能実現を優先しているため，教員の作成したテンプレートから生成されたプログラムや学生の提出プログラム実行時のセキュリティは考慮されていないが，それらの登録時の危険コード除去や chroot などを使用した実行環境制限は今後検討していく必要がある．

**謝辞**　本研究は科研費（課題番号 16K01022）の支援を受けています．

## 参考文献

[1] John Allspaw，Paul Hammond，10+ Deploys Per Day: Dev and Ops Cooperation at Flickr，2009.
[2] 日経 SYSTEMS 編集，Jenkins、Chef、Redmine、Docker で業務効率アップ：10 倍速の開発・運用ツール（日経 BP ムック），日経 BP 社，2015.
[3] 大田和樹，高崎光浩，大月美佳，掛下哲郎, DevOps ツールを活用したソフトウェア開発技術者教育支援システムの構想，情報処理学会 第 135 回情報システムと社会環境研究発表会，2016-IS-135(4), pp.1-8, 2016.
[4] 和田貴久、河村雅人、米沢弘樹、山岸啓，改訂新版 Jenkins 実践入門，技術評論社，2015.
[5] Jon Loeliger，実用 Git，オライリージャパン，2010.
[6] Steve Holzner，Ant 第 2 版，オライリージャパン，2005.
[7] Checkstyle， Checkstyle 6.12.1 : http://checkstyle.sourceforge.net/
[8] FindBugs, Find Bugs in Java Programs: http://findbugs.sourceforge.net/
[9] 渡辺修司，JUnit 実践入門，技術評論社，2012.
[10] Kent Beck，テスト駆動開発入門，ピアソンエデュケーション，2003.
[11] 小西達裕，鈴木浩之，伊東幸宏, プログラミング教育における教師支援のためのプログラム評価機構，電子情報通信学会論文誌，Vol.J83-D-I，No.6，pp.682-692, 2000
[12] 蔵本 幸司, 西村 智治, 富永 浩之, 小コンテスト形式の初級 C 演習における教師支援－実行テスト系列に即したプログラミング問題のオーサリングツール，情報処理学会 第 115 回コンピュータと教育研究会, 2012-CE-115(7), pp.1-6, 2012
[13] CSUS Programming Contest Control (PC^2) Home Page , http://pc2.ecs.csus.edu/