Vol.2016-IS-135 No.4
2016/3/7

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

大田和樹[†1]　高崎光浩[†2]　大月美佳[†3]　掛下哲郎[†3]

概要： 本論文では各種の 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 を提案する．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．本稿では，システムの全体構想と本システムを実際の授業で使うための計画および準備状況を示す．

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

Concept of Software Engineer Education Support System utilizing DevOps Tools

Kazuki Ota[†1]　Mitsuhiro Takasaki[†2]　Mika Ohtsuki[†3]　Tetsuro Kakeshita[†3]

Abstract: We propose the concept of software development engineer education support systems named ALECSS utilizing various DevOps tools. ALECSS automatically checks the student programs from various perspectives and returns feedbacks to the students. The system thus facilitates students to learn techniques to develop high-quality software. The system also supports evaluation process of the student programs. ALECSS performs various types of checking such as compilation checking, checking of execution results and programming style, and static code analysis. The system is developed utilizing various DevOps tools such as continuous integration tool Jenkins and software configuration management tool Git. In this paper, we show the entire design of ALECSS along with the plan and preparation status for use of the system at an actual lecture.

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．しかし，チーム開発を進めていくと，チームは様々な問題に直面する．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もある．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]を活用した自動化が進んでいる．我々は，これらの DevOps ツールを教育目的で活用する．

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，学生が提出したコードは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．同様の授業は，プログラミング教育等で多数見られる．

そこで，本論文では，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）を提案する．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．

本論文では，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する．両者は，教員および学生の教育・学習ワークフローを最適化するように設計されている．

本論文の 2 節では，システム開発実験の概要と ALECSS に対する要求事項について説明する．3 節では ALECSS の全体構想を説明する．4 節以下では ALECSS を構成する DevOps ツールの設定について説明する．4 節では継続的インテグレーションツール Jenkins [3]，5 節では，バージョン

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

管理ツール Git [4]，6 節では，ビルドツール Ant [5]の設定についてそれぞれ説明する．7 節では，学生が提出したプログラムを様々な観点から点検するために，プログラミングスタイルチェッカーCheckstyle [6]，プログラム静的解析ツール FindBugs [7], ユニットテストツール JUnit [8]の設定を示す.

2. 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基本的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（初期は CVS，現在は Git），統合開発環境 Eclipse，単体テストフレームワーク JUnit（初期はバージョン 3，現在はバージョン 4）およびツリー型の掲示板（進捗管理用）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．

個人演習では，インストール回の後，Git 演習（1 回），Java 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．

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．

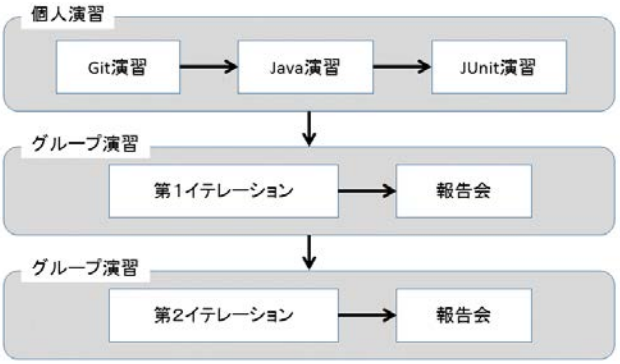

図 1　授業概要
Figure 1　Course Outline

3. ソフトウェア開発技術者教育支援システム ALECSS

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．ソフトウェア開発技術者教育支援システム ALECSS の全体構想を図 2 に示す．ALECSS は，これらの評価項目のチェックを可能な限り自動化するように構想されている．

学生が作成したソースコードは統合開発環境（IDE：Integrated Development Environment）Eclipse と連携したバージョン管理ツール（VCS: Version Control System）の Git を

表 1　システム開発実験における評価項目と自動チェック計画
Table 1 Evaluation Criteria and Automatic Evaluation Plan for Experiment on System Development

演習課題	評価項目	自動チェック計画
Git 演習	課題ファイルが全て提出されているか	Ant を活用（ファイルやログを取得して望ましい結果と比較する独自スクリプトを用意）
	指定した作業を全て行っているか	
Java 演習	プログラムがコンパイルできるか	Ant の実行結果を確認
	コーディング標準に準拠しているか	Checkstyle を活用
	提出された構成が正しいか	Ant を活用（独自スクリプトを用意）
	プログラムの出力結果は正しいか	
JUnit 演習	全てのテストが成功するか	JUnit 実行プラグインを活用
	内容が実装されているか	Ant を活用（独自スクリプトを用意）
	テストコードは妥当か	Junit を活用（テストコードをテストするコードを用意）
グループ演習	典型的な落とし穴に陥っていないか	FindBugs を活用

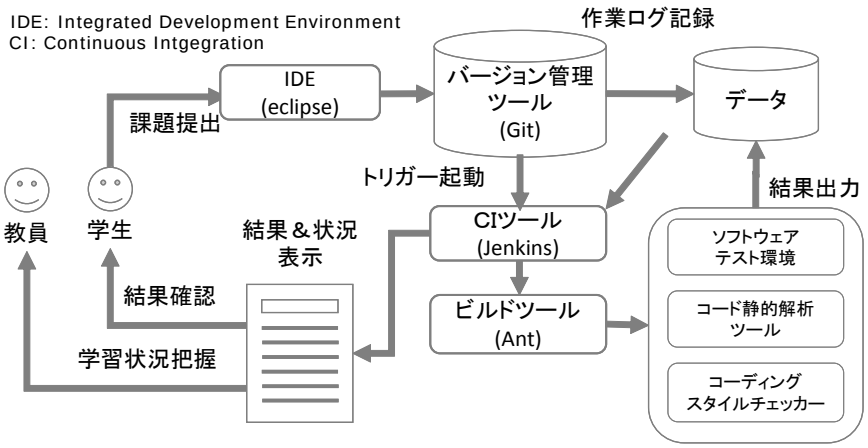

図 2　システムの全体構想

Figure 2　Entire Structure of the System

介して提出される．ソースコード更新通知を受けた継続的インテグレーション（CI : Continuous Integration）ツールは構成管理ツール（Ant, Maven など）に実行指示を出す．構成管理ツールは，ビルドを行うとともに，プログラミングスタイルチェッカー（Checkstyle 等），コード静的解析ツール（FindBugs 等），ユニットテストツール（JUnit 等）といったプログラムチェッカーに実行指示を出す．CI ツールは，ビルド等の結果を各ツールから取得する．取得した結果を学生にフィードバックする．これと同時に，学生がビルドを行った結果や進捗状況を教員が閲覧でき，学生の評価に利用できるようにする．

提出したコードは，個人演習では作成した学生自身，グループ演習では作成したグループの学生しか見ることができないようにする必要がある．

ツール間の連携を実現するには, CI ツールである Jenkins に各ツールに対応したプラグインをインストールする必要がある．プラグインをインストールすると Jenkins 上で各ツールの設定を行える．

4. CI ツール Jenkins の設定

CI ツールは，バージョン管理ツール，ビルドツール，各種のチェックツール等を組み合わせ，継続的インテグレーション作業を行うためのツールである．本システムでは，CI ツールとして Jenkins を使用する．

4.1 ジョブ設定

Jenkins では，ビルドに必要な JDK や Ant を自動的にインストールすることができる．この機能は，トップページにある「Jenkins の管理」に移動し，「システムの設定」画面から行うことができる．

Jenkins でビルドを実行するためには，ジョブを設定する必要がある．トップページの「新規ジョブの作成」を選択すると新しいジョブの作成画面が表示される（図 3）．

図 3　新規ジョブ作成画面

Figure 3　Creation of New Job

ジョブ名には今から作るジョブの名前を入力し，「フリースタイル・プロジェクトのビルド」を選択し，OK ボタンをクリックする．ここまでで新規ジョブの作成が完了した．

次にビルドに必要な設定を行う．始めにバージョン管理ツールの設定を行う．

ALECSS では Git を使用するが, デフォルトの状態では, 選択できない. Git を選択できるようにするには, Git Plugin をインストールする必要がある．Git の設定ではまずビルドに必要なソースコードを格納している Git のリポジトリパスをリポジトリ URL に設定する. 指定したリポジトリパスによっては認証情報が必要となるため，必要に応じて認証情報を設定する．どのブランチをビルドするかに関してはデフォルトでは master が指定されている．リポジトリ・ブラウザではビルド間の差分を表示するためのブラウザを選択することができる．

次に実行するビルド処理の設定を行う．Jenkins でのビルド処理では, Ant や Maven といったビルドツール, Windows バッチコマンドおよび Unix のシェルコマンドが利用できる．ここでは，Ant でビルドを設定していく（図 4）．

図 4　ビルド設定画面
Figure 4　Build Settings

Ant では，5 つの項目の設定を行う．ターゲットでは，Ant で実行するターゲットのリストを指定する．指定がない場合は，ビルドファイルのデフォルトターゲットを実行する．ビルドファイルでは，実行するビルドファイルを指定する．何も指定がない場合は，ルートディレクトリの build.xml を実行する．プロパティでは，Ant のビルドで必要なプロパティを作成する．Java オプションは，Ant のオプションが必要なときに指定する．

最後にビルド後に実行する処理を設定する．実行できる処理は図 5 の通りである．Checkstyle 警告の集計，FindBugs 警告の集計，Git publisher は，プラグインをインストールすることで追加された処理である．

図 5　ビルド後の処理の追加設定画面
Figure 5　Additional Settings for the Processing after Build

4.2 権限管理

Jenkins では，ジョブ（提出物）の一覧が表示されているが，Jenkins にアクセスできる利用者ならば誰でもこの一覧が閲覧できる状態にある．個人が出した課題は提出者自身，チームで提出した課題であれば，チームメンバのみが閲覧できる状態でなければならない．また教員は，提出されたすべての課題を閲覧できなければならない．そのためには，提出した課題に対して適切な権限の設定を行わなければならない．

権限の管理を行うプラグインとして，Role Strategy Plugin がある．このプラグインを用いると，Global Roles と Project Roles を設定できる．Global Roles は，すべてのジョブや Jenkins 自体の設定に関係する権限である．Project Roles は，特定のジョブに限定した権限を表現する．

Global Roles では，教員には，すべての権限を与えた Admin 権限を付与し，学生には，ジョブの一覧を閲覧できる Read 権限ジョブを作成する Create 権限を与える必要がある．

Project Roles に関しては，教員に対しては，すべての権限を与える Admin 権限を付与し，学生に対しては，ジョブ提出にかかわったメンバのみに課題を提出するために必要な権限を付与し，その他の学生に対しては閲覧もおこなえない設定とする．

このような設定を行うことで，教員には，すべての権限を持たせることができる．また，学生は，ジョブの一覧を確認できるが，詳細は確認できず，自身が関わったジョブにのみ権限を持つことができる．現状では，権限の付与を手動で行う必要があるが，将来的には，権限の付与作業を自動化する予定である．

5. バージョン管理ツール Git の設定

バージョン管理ツールとは，あるファイルをいつ・誰が・どのように変更したのかという履歴をバージョンとして記録して管理するツールである．本システムでは，バージョン管理ツールとして Git を使用する．Jenkins で Git を利用するためには，Jenkins に Git Plugin をインストールする必要がある．「Git Plugin」のインストールが完了すると Jenkins の「システムの設定」画面に「Git」の欄が追加される（図 6）．

図 6　Git 設定画面
Figure 6　Git Settings

ここでは，「Name」は，任意の名前を「Path to Git executable」には Git の実行ファイルのパスを設定する必要がある．

「ジョブの設定」画面では，前述の通り，「ソースコード管理」の欄に Git が追加される．

また，「ビルド後の処理」の欄で「Git Publisher」を追加するとビルド後に実行する Git リポジトリへの処理を行うことができる（図 7）．

図 7　Git Publisher 設定画面
Figure 7　Git Publisher Settings

ここでは，ビルド成功時のプッシュ・マージ・タグの作成，ブランチ作成，ノート作成（コミットメッセージで書けなかった注釈を，コミット履歴を変更せずに追加する機能）の処理を設定できる．

6. ビルドツール Ant の設定

ビルドツールとはプログラムを構築するための一連の作業を自動化するもので，あらかじめ記述したルールに従いコンパイラ等を呼び出すツールである．本システムでは，ビルドツールとして Ant を使用する．Ant は Jenkins による自動インストールができる．Ant でビルドを実行するためには，まず何をどのようにするのかを XML 形式で記述した「build.xml」というファイルを用意する必要がある．

「build.xml」ファイルは，大きく分けると「プロジェクト」「ターゲット」「タスク」「プロパティ」の 4 つの要素から構成される．

プロジェクト要素は build.xml ファイルのトップレベルの要素である．プロジェクト要素は，プロジェクトの名前，デフォルトで実行するターゲット，パス指定のベースとなるディレクトリの設定を行う．

ターゲット要素はタスクの集まりをまとめたものである．複数のターゲットを 1 つのプロジェクト内で記述することができる．Ant でビルドするときは，タスクごとに実行するかどうかを設定する．

タスク要素は実行する具体的な処理を記述する．ファイルをコピーする，コンパイルするなどの実際に行う処理を行う．タスクはターゲット要素内に記述され，複数のタスクを 1 つのプロジェクト内に記述することができる．

プロパティ要素は，build.xml 内で使用される値を定数として定義するために使用する．

作成した build.xml は，プロジェクトのリポジトリ内に保存する必要がある．Ant では，ビルドだけではなくプログラムチェッカーの実行指示も行う．ユニットテストツールの JUnit を実行させる junit タスクは Ant 側で用意されている．しかし，プログラミングスタイルチェッカーである Checkstyle とプログラム静的解析ツールである FindBugs に関しては，実行させるタスクが Ant 側に用意されていない．そのため，プログラムチェッカー側が用意しているタスクを Ant にロードさせて，checkstyle タスク，findbugs タスクを作成する必要がある．taskdef タスクというタスクを作成するタスクを build.xml に追加することでプログラミングスタイルチェッカー側が作成したタスクを読み込み，新しいタスクを作成できる（図 8）．タスク作成に必要なファイルとして Checkstyle では，「checkstyle.jar」，FindBugs では，「findbugs-ant.jar」が必要となる．

```
<target name="maketask" >
 <taskdef
resource="com/puppycrawl/tools/checkstyle/ant/checks
tyle-ant-task.properties"
classpath="lib/checkstyle-6.12.1/checkstyle.jar"/>
 <taskdef
name="findbugs"
classname="edu.umd.cs.findbugs.anttask.FindBugsTas
k"
classpath="lib/findbugs/findbugs-ant.jar"/>
</target>
```

図 8　taskdef タスク記述例
Figure 8 Task　Description Example using taskdef

6.1 Ant による評価項目の設定

Java 演習でチェックする「ファイル構成が正しい」という項目は，condition タスクを使用することで実現できる．condition タスクは指定された条件が真の場合，プロパティを設定するタスクである．条件が真でない場合，プロパティは設定されない．

ファイルチェックの実行例では，まず，確認したいファイルの場所を mytest プロパティで設定する（./src/Main.java）．次に condition タスクで，条件が真なら OK という値を格納した check プロパティを作成すると設定する．available 要素では，指定されたファイルが，存在すれば真を返す．最後に，echo で結果を出力する．プロパティ設定ができていれば，value で設定した内容，デフォルトでは true が返ってくる．

```
<target name="check">
 <property name="mytest" value="./src/Main.java"/>
 <condition property="check" value="OK">
  <available file="${mytest}"/>
 </condition>
 <echo message ="${check}"/>
</target>
```

図 9　ファイルチェック記述例
Figure 9　Example of File Checking

```
check:
     [echo] OK

     BUILD SUCCESSFUL
```

図 10　実行結果
Figure 10　Execution Result of File Checking

Java 演習でのチェックする「プログラムの出力結果が正しいか」という項目は，Ant からスクリプトを呼び出して diff による一致判定を用いることで実現できる．

一致判定の実行例として Ant 側では，学生が作成したプログラムの実行結果をテキスト形式で保存する．次に保存されたディレクトリを Project_HOME としてプロパティを作成する．最後にスクリプトを実行結果テキストファイルの絶対パスを引数として実行している．スクリプト側では，あらかじめ作成しておいた正解が記述されたテキストファイルと実行結果テキストファイルを比較している．両者が完全に一致していれば，「一致しています」を，異なっていれば「一致していません」を出力する．出力結果は，Ant 側で指定したアウトプットファイルに保存される．

```
<target name="java">
  <java classname ="Main" classpath="./class"
    output="result.txt"/>
  <dirname property="Project_HOME" file="ant.file"/>
  <apply executable="${Project_HOME}/answer.sh"
 dir="${Project_HOME}" output="evaluation.txt">
  <fileset file="${Project_HOME}/result.txt"/>
  </apply>
</target>
```

図 11　出力結果の一致判定（Ant）
Figure 11　Matching of Output (Ant)

```
#!/bin/sh
if diff answer.txt $1 ; then
echo "一致しています"
else
echo "一致していません"
fi
```

図 12　出力結果の一致判定用スクリプト
Figure 12　Script for Output Matching

7. プログラムチェッカーの設定

7.1 プログラミングスタイルチェッカーCheckstyle

プログラミングスタイルチェッカーとは，プログラムを実行せずにソースコードのコーディング規約違反の可能性のあるコードを検出するツールである．本システムでは，プログラミングスタイルチェッカーとして Checkstyle を使用する．Checkstyle は現在，Sun Code Conventions と Google Java style の 2 つの Java コーディング標準をサポートしている．プロジェクト独自のルールで行う場合は，規約ファイルを別途用意する必要がある．

Checkstyle を利用するときは，checkstyle.jar を配布ページからダウンロードし，プロジェクトのリポジトリ内に保存しておく必要がある．また，Checkstyle の検出結果を Jenkins で記録したいときは，Jenkins に「Checkstyle Plug-in」をインストールする必要がある．プログラミングスタイルチェックをするときは，Ant を実行のために必要な build.xml 内に Checkstyle に関する記述を加える必要がある．図 8 にあるように taskdef タスクで checkstyle タスクを使用できる状態にした上で記述する必要がある（図 13）．

```
<target name="checkstyle" >
  <checkstyle config="./lib/sun_checks.xml"
        failOnViolation="false">
  <fileset dir="./src" includes="*.java"/>
  <formatter type="xml"
        tofile="./report/checkstyle_report.xml"/>
  </checkstyle>
</target>
```

図 13　checkstyle タスク記述例
Figure 13　Task Description Example for Checkstyle

「Checkstyle Plug-in」をインストールすることで「ビルド後の処理」の欄で「Checkstyle 警告の集計」を追加することができる．追加すると Checkstyle が作成した集計ファイルを読み込んで集計が行える（図 14）．

図 14　Checkstyle 警告の推移画面
Figure 14　Transition of Checkstyle Warning

7.2 Checkstyle を利用した評価項目設定

Java 演習では，「コーディング標準に準拠しているか」という評価項目が存在する．その項目に対して Checkstyle を活用する．本システムでは，コーディング標準として Sun Code Conventions を使用する．このコーディング標準は，Checkstyle.jar 内に sun_checks.xml として定義されている．

チェックするファイルは，提出されたすべての java ファイル，チェック結果は XML 形式で出力する．

出力されたチェック結果は，警告の総数の他に前回のビルドから減った警告数と前回から増えた警告数，警告の詳細が閲覧できる．この警告の総数や警告の詳細を活用して評価を行うこともできる．

7.3 プログラム静的解析ツール FindBugs

プログラム静的解析ツールとは，ソースコードやバイトコードの文法チェックおよび意味的チェックをコンパイラよりも厳格に行い，コードの誤りを事前に検出することで，ソフトウェア品質を高めるためのツールである．本システムでは，プログラム静的解析ツールとして FindBugs を使用する．

FindBugs を利用するときは，プログラムファイルを配布ページからダウンロードし，圧縮されている場合は回答した上で任意のフォルダ内に保存しておく必要がある．また，FindBugs の検出結果を Jenkins で記録したいときは，Jenkins に「FindBugs Plug-in」をインストールする必要がある．プログラム静的解析を行うためには Ant の実行に必要な build.xml 内に FindBugs に関する記述を加える必要がある．図 14 にあるように taskdef タスクで findbugs タスクを使用できる状態にした上で記述する必要がある（図 15）．

```
<target name="findbugs" >
  <findbugs home="home/user/findbugs-3.0.1"
    output="xml"
outputFile="./report/findbugs_report.xml">
  <class location="./class" />
  </findbugs>
</target>
```

図 15　findbugs タスク記述例
Figure 15　Task Description Example for FindBugs

「FindBugs Plug-in」をインストールすることで「ビルド後の処理」の欄で「FindBugs 警告の集計」を追加することができる．追加すると FindBugs が作成した集計ファイルを読み込んで集計が行える．

7.4 ユニットテストツール JUnit

ユニットテストツールとは，テストを自動的に実行するツールである．本システムでは，ユニットテストツールとして JUnit を使用する．

JUnit を利用するときは，junit.jar を配布ページからダウンロードし，プロジェクトのリポジトリ内に保存しておく必要がある．JUnit のテスト結果を Jenkins で記録するためのプラグインは Jenkins インストール時に同梱されているためインストール不要である．ユニットテストを行うためには Ant の実行に必要な build.xml 内に JUnit に関する記述を加える必要がある（図 16）．Ant バージョン 1.7 以降では，junit タスクの実行に必要な ant-junit.jar がバイナリに同梱されているため，Checkstyle や FindBugs のようなタスクの作成は不要である．

```
<target name="junit" >
  <junit>
  <classpath>
  <fileset dir="lib" includes="junit.jar" />
  </classpath>
  <test name="example.Test" />
  </junit>
</target>
```

図 16　junit タスク記述例
Figure 16　Task Description Example for JUnit

「ビルド後の処理」の欄で「JUnit テスト結果の集計」を追加することができる．追加すると JUnit が作成したテスト結果ファイルを読み込んで集計が行える（図 17）．

図 17　JUnit テスト結果の集計設定
Figure 17　Setting for JUnit Test Results Summary

本来 JUnit のテストを書く時は，まずそのテストを失敗する実装を用意しておき，書いたテストコードが失敗するのを確認後，成功するようにコードを書き換えるという手順を取る[9]．しかし，慣れない初学者はこの過程を行わずにテスト対象の実装を書いた後で，それに対するテストコードを書くことがある．後からテストコードを書くと，正しくない実装に対してテストが失敗することが保証されない．実際，間違った実装に対しても成功してしまうようなテストコードが書かれることがある．このようなテストコードになっていないかを自動チェックすることを考える．

具体的には，テストが失敗するような実装を用意しておき，学生の書いたテストコードでその実装をテストして失敗することを確認できるようにする．例えば，3 の倍数に対して"Fizz"，5 の倍数に対して"Buzz"，3 と 5 の倍数に対して"FizzBuzz"，それ以外の場合には数値を文字列にしたものを返す FizzBuzz のメソッドをテストする場合，その実装を null や"boo"のような間違った文字列を返すようにしておけば，4 つの場合についてのテストはすべて失敗するはずである．

このように全てを通すようなテストを検出するため，教員側が間違った実装コードを用意しておき，そこへ学生の提出したテストコードをコピーしてきてテストを行い，失敗することを確認できるような仕組みを用意したい．例えば，FizzBuzz で言えば，本来は図 18 のようなテストコードであるべきだが,学生が図 19 のようなテストコードを書いた場合，テストは常に成功してしまう．これに対して，図 20 のような意図的に誤りを実装したコードを教員が用意することで，常にテストが成功するテストコードを検出できる．

```
@Test
public void testFizz() {
   FizzBuzz sut = new FizzBuzz();
   assertThat(sut.check(3), is("Fizz"));
}
```

図 18　妥当なテストコード例
Figure 18　Example of Valid Test Code

```
@Test
public void testFizz() {
   assertThat(3, is(3));
}
```

図 19　妥当でないテストコード例
Figure 19　Example of Invalid Test Code

```
public String check(int x) {
   return null;
}
```

図 20　妥当でないテストコードを検出するコード例
Figure 20　Sample Code to Detect Invalid Test Code

手順としては以下のようになる．

1. 教員はあらかじめテストに対して間違った実装を登録しておく
2. 学生が対応するテストコードを提出した段階で，間違った実装とそのテストを作業領域にコピーする
3. それらを自動テストし，学生ごとにログとして記録する
4. ログをチェックし結果が失敗していない場合は警告が表示されるようにする

8. おわりに

本論文ではソフトウェア開発技術者教育支援システム ALECSS の構想と，ALECSS を授業で使うための計画と各種 DepOps ツールの設定方法について述べた．

ALECSS を導入すると，学生が作成した課題を提出することで，自動的にビルドを実行するとともにテストやコーディング標準のチェック，静的コード解析を行い，その結果をフィードバックする．学生は，ある程度の成果物を ALECSS に提出することで，教員や TA に頼ることなく成果物の問題点や未達点を確認できるようになる．教員は，学生がその日の授業で最後に提出したときのフィードバックを閲覧することで，学生の進捗状況や理解度を確認できるとともに成績評価の判断材料を得ることができる．

今後は,今回立てた計画に基づいて ALECSS を完成させ，授業で運用できる状態にする．また，運用後は ALECSS で得たデータや利用者の意見を集めて分析し，学生の理解度分析やシステムの改良を進める予定である．現状では，Jenkins や Git のユーザ登録はツールごとに行わなければならない. 大学では Moodle [11]等の LMS（Lecture Management System）の活用が進んでいるため，Moodle で登録されたユーザデータを用いて Jenkins や Git のユーザ登録が自動化できるように改良したい．

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] 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.
[4] Jon Loeliger，実用 Git，オライリージャパン，2010.
[5] Steve Holzner, Ant 第 2 版, オライリージャパン, 2005.
[6] checkstyle – Checkstyle 6.12.1 : http://checkstyle.sourceforge.net/
[7] FindBugs™ - Find Bugs in Java Programs: http://findbugs.sourceforge.net/
[8] 渡辺修司，JUnit 実践入門，技術評論社，2012.
[9] Kent Beck，テスト駆動開発入門，ピアソンエデュケーション，2003.
[10] Kent Beck、 Cynthia Andres，エクストリームプログラミング，オーム社，2015.
[11] 井上博樹，Moodle 2 ガイドブック―オープンソースソフトウェアでオンライン教育サイトを構築しよう，海文堂出版，2013.