Utilizing Software Engineering Education Support System ALECSS at an Actual Software Development Experiment: A Case Study

Mika Ohtsuki and Tetsuro Kakeshita
Department of Information Science, Saga University, Saga, Japan

Keywords: Software Engineering Education, DevOps Tool, Software Verification, Coding Style Checking, Execution Checking, Static Checking, Peer Review.

Abstract: We have proposed a software engineering education support system named ALECSS in our previous paper. ALECSS utilizes various DevOps tools such as Jenkins, Git, JUnit, Checkstyle and FindBugs to automatically check student's programs from various viewpoints and to quickly provide feedbacks to the students. At the same time, ALECSS collects student's log so that a teacher can easily observe the status of each student and/or each project team to improve software engineering education. In this paper, we utilize ALECSS at an actual software development experiment for self and peer review of the source code. Students are grouped into project teams and each student can view summary pages for the student or the team containing the messages generated by the DevOps tools integrated into ALECSS. We also collected feedback from the students and received many positive comments.

1 INTRODUCTION

It is quite important to utilize contemporary software development tools to realize practical software engineering education (Nandigam, 2008). It is also important to teach management of quality, cost and delivery (QCD) of software development. Contemporary software engineering education requires collaboration of students as a team since the software size is typically too large to work on a student.

We are teaching collaborative software development at the third academic year of our department majored in computer science. In our class, a student team faces with many problems. For example, it is hard to see and control progress of the software development project without appropriate sharing of the known problems and progress of the software development tasks at each source code among team members. It is also necessary to ensure QCD of the software. In order to cope with such problems, realistic software developers are shifting to utilize various DevOps tools (Allspaw, 2009; Bass, 2015).

In our previous paper (Ohtsuki, 2016), we proposed a software engineering education support system named ALECSS (Automated Learning and Evaluation Cycle Support System). ALECSS automatically checks source codes submitted by a students and/or a team from various viewpoints and returns feedbacks to them. Various DevOps tools, such as JUnit, Git, Ant, Checkstyle, FidBugs and Jenkins, are utilized for the checking and for the integration of the checking tools. We also added original scripts to ALECSS for the checking of test code and Git working status. A student or a team can check their codes quickly and can improve them promptly by utilizing ALECSS. At the same time, the teacher can easily observe progress of the students and the team.

We utilized ALECSS to an actual software development experiment in the 2018 spring semester. In this paper, we report the results of the experimental application. Although the students use ALECSS for the first time, their evaluation of ALECSS is quite good.

In Section 2, we shall explain the major functions of ALECSS. We next explain the software development experiment in Section 3 and how to utilize ALECSS in the experiment in Section 4. We present and discuss the result of our evaluation in Sections 5 and 6. In section 7, we show the related works and compare them with our contribution.

Ohtsuki, M. and Kakeshita, T.
Utilizing Software Engineering Education Support System ALECSS at an Actual Software Development Experiment: A Case Study.
DOI: 10.5220/0007723203670375
In *Proceedings of the 11th International Conference on Computer Supported Education (CSEDU 2019)*, pages 367-375
ISBN: 978-989-758-367-4

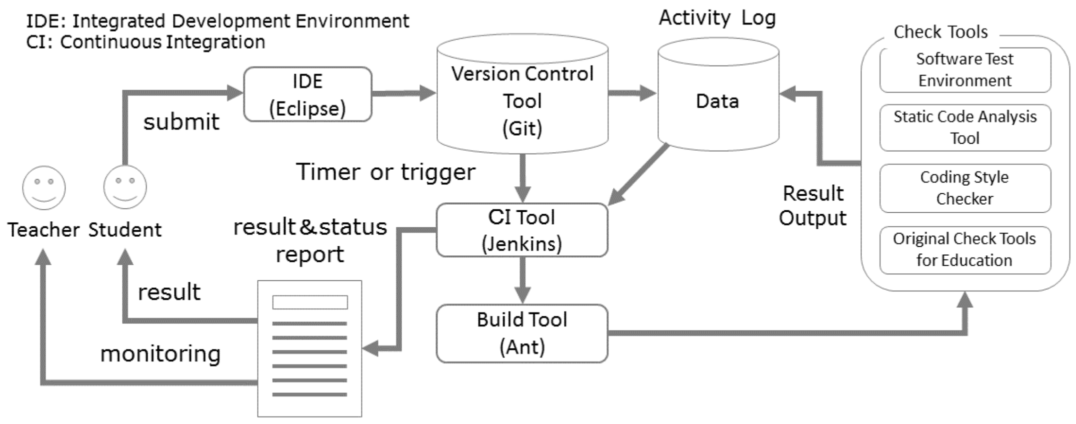


Figure 1: Entire structure of ALECSS.

2 SOFTWARE ENGINEERING EDUCATION SUPPORT SYSTEM ALECSS

ALECSS is developed using various DevOps tools such as Git, Jenkins, Ant, Checkstyle, FindBugs and JUnit as well as additional scripts for further checking of the student's code. Figure 1 illustrates the entire structure and the behaviour of ALECSS.

A student can edit and submit a source program to ALECSS by uploading the file(s) to the Git repository from the integrated development environment Eclipse. Then Git notifies the file uploading to Jenkins. Jenkins automatically starts the build tool Ant by utilizing the submission as a trigger. Ant is controlled by the configuration file build.xml which contains setup and execution commands of the various DevOps tools. The checking results are collected by Jenkins at the activity log. Then the student and student team can browse the checking result and the status report to improve their source code. The teacher can also browse the same report to understand status of each project.

Various types of evaluation criteria can be automatically checked using ALECSS. The criteria and the implementation of the checking mechanism are explained at the rest of this section.

2.1 File Structure Checking

We can perform the existence checking of the required files by using condition tag and available tag defined as Ant tasks. File Structure Checking is required for all exercises and the file names are different depending on the exercise. Furthermore, file and folder names are assigned depending on the student number or project name at several exercises. Therefore, it is necessary to have scripts to generate appropriate names from student number and/or project name. Thus we can generate build.xml utilizing the scripts for each exercise of the experiment.

2.2 Coding Standard Checking

Coding standard checking ensures that the submitted Java program keeps one of the default coding style definitions (Sun Code Conventions and Google Java style). The checking is performed by executing coding style checker Checkstyle. Execution of Checkstyle is defined as an Ant task by using the taskdef clause in build.xml.

2.3 Compile Checking

Compile checking is a prerequisite of all other checking. ALECSS executes a standard compilation by invoking the javac command as an Ant task and can show the compilation result of the submitted Java program. If the compilation fails, students can browse the error messages on Jenkins.

2.4 Output Result Checking

The output result checking is performed to ensure that the output of the submitted program matches to the output defined by the specification. ALECSS executes a compiled Java program and record the output to the log which can be observed on Jenkins. Furthermore, we prepare a script to compare the output log with a predefined file.

Such script can be implemented by using the diff command if the result is fixed. For the case that the output depends on student number, student name or project name, we develop a special script using the same technique as explained for File Structure Checking.

2.5 Git Work Execution Checking

Git maintains a commitment log storing four types of Git actions: file addition (add and commit), file deletion (remove and commit), file update (edit and commit), revert (return to a former commitment state). The Git work execution checking examines the Git log to confirm that a student correctly performs the required Git operations. The commitment log can be accessed by the Git log command. We are developing a script for the Git work execution checking.

2.6 JUnit Execution Checking

JUnit is utilized to check whether the subroutines in the submitted program return correct values. Teachers need to provide a set of test codes executed by JUnit for the checking. JUnit can be executed as an Ant task with a junit tag. We can obtain the log using the task and can observe the number of successes/failures of the unit test on Jenkins.

2.7 Test Case Null Implementation Checking

Our experiment also contains exercises to develop test code. If a student develops an empty test code, any test will succeed in the JUnit framework. The test case null implementation checking detects such empty test cases. This can be implemented by counting the number of lines in each test case method. We have developed a Java language parser utilizing JavaCC to extract a test case method and a script counting the number of lines of the test case method (Koga, 2018). The parser and the script are defined as Ant tasks for automatic execution.

2.8 Test Code Validation

The test code validation is executed to detect incorrect test code which succeeds for any input. In order to implement such validation, we prepare project code which all tests fail. The project codes are copied to the working area of the student in order to confirm that the test code developed by the student correctly fails for the project code.

2.9 Static Code Checking

Static code analysis tool FindBugs is utilized for the checking to detect pitfalls which can be observed within a Java source code. Execution of FindBugs is defined as an Ant task as in the case of Checkstyle.

3 COOPERATIVE SOFTWARE DEVELOPMENT EXPERIMENT

Our cooperative software development experiment is provided for the undergraduate student at the third academic year. The experiment is a compulsory subject for graduation and usually about 60 students are enrolled to the experiment each year. Our department is accredited as a computer science program and the students have learned software engineering and basic Java programming before the experiment. The experiment consists of fifteen weeks of 3 hour exercises. Table 1 represents the experiment plan.

Table 1: Experimental plan.

Week	Description
1	Setting up software development environment (Git and Eclipse)
2	Git Exercise
3	Java Exercise (including Checkstyle and Javadoc)
4-5	JUnit Exercise
6	Introduction to Group Exercise (Group formation, Ice Breaking and Explanation of Requirements)
7-9	Implementation (First Iteration) Introduction to ALECSS (at Week 9)
10	Peer Review (First Iteration)
11-13	Explanation of Additional Requirements Implementation (Second Iteration) Student Survey (at Week 12)
14	Peer Review (Second Iteration)
15	Bug Fix and Second Student Survey

Students work on individual exercises at the first 5 weeks. We introduce Git, Eclipse, Checkstyle, Javadoc and JUnit during these weeks.

The group exercises starts at week 6. Group exercises are carried out using the baseline project initially distributed to the 8 student teams (Cherry, Dandelion, Lily, Peach, Plum, Rose, Sunflower, and Violet). Each team consists of 7-8 students. The group exercise consists of two iterations. Each of the iterations contains three weeks for implementation exercise and one week for peer review by other students.

We applied ALECSS to the experiment in 2018. At first, we introduce ALECSS at week 9 so that students start using ALECSS from the week. At week 9, student teams can only check their own project using ALECSS. However they can also utilize ALECSS to check projects developed by other teams at week 10 by allowing access to other project. Then at the end of the experiment, student teams utilize ALECSS for the second peer review (at week 14) in the second iteration. We performed the second student survey and collected the final project data after week 15 where detected bugs are expected to be fixed.

In the peer review, a group reviews another team's project assigned randomly by the teacher and reports review result containing detected issues. The types of issues required to be reported are coding standard violations, portion of algorithms which are not bugs but contain some problems, software bugs or unimplemented requirements. The reported issues are checked by the teachers whether they are reasonable. A team gets some score when the team reports reasonable issues and the team loses the same score when the issue is reported. The score for each issue is defined as follows depending on the importance of the issues.

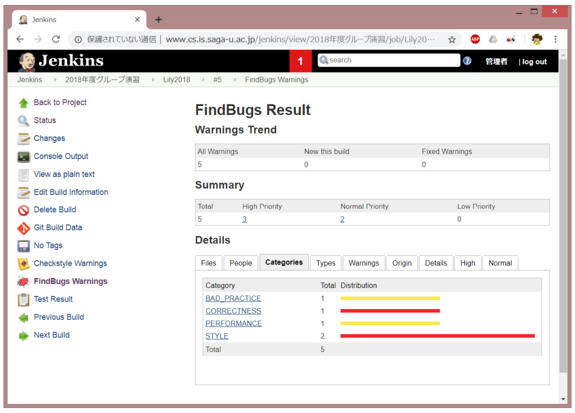
- 1/3 : Coding Style Violation
- 1-3: Problems in Algorithm
- 2-5: Software Bugs and Unimplemented Specification

The number of issues which can be reported by a team is at most twenty so that the score to be earned or lost is bounded.

4 UTILIZING ALECSS AT THE EXPERIMENT

ALECSS can be executed either automatically or manually. We have already explained the automatic execution in Section 2. A student or a team can also execute ALECSS manually after selecting a project. The manual checking function is used at the peer reviews.

Figure 2 represents a result of static code checking. The warning messages generated by FindBugs are classified by categories. The red bars represent the number of warnings with high priority, while the yellow bars represent the number of warnings with normal priority. Each category has a priority represented by the color of the corresponding bar.



FindBugs Result

Warnings Trend

All Warnings	New this build	Fixed Warnings
5	0	0

Summary

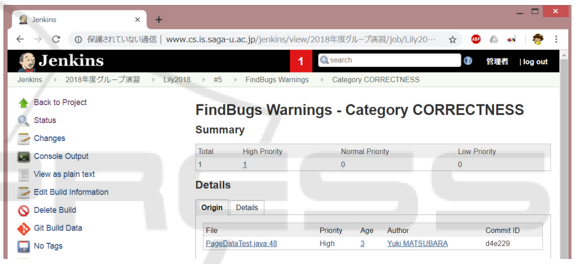
Total	High Priority	Normal Priority	Low Priority
5	3	2	0

Details

Category	Total	Distribution
BAD_PRACTICE	1	🟨
CORRECTNESS	1	🟥
PERFORMANCE	1	🟨
STYLE	2	🟥
Total	5	

Figure 2: Result of static code checking.

When a student selects a category in Figure 2, the warnings belonging to the selected category are shown (Figure 3). Each warning is represented by the class name and the line number at the File column. Priority, author and commit ID are also represented.


Jenkins
保護されていない通信 | www.cs.is.saga-u.ac.jp/jenkins/view/2018年度グループ演習/job/Lily20…
Jenkins
1
search
管理者
| log out
Jenkins › 2018年度グループ演習 › Lily2018 › #5 › FindBugs Warnings › Category CORRECTNESS
Back to Project
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete Build
Git Build Data
No Tags
Origin
Details

FindBugs Warnings - Category CORRECTNESS

Summary

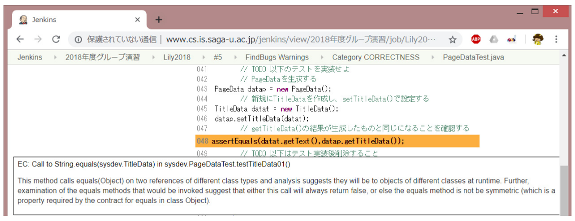
Total	High Priority	Normal Priority	Low Priority
1	1	0	0

Details

File	Priority	Age	Author	Commit ID
PageDataTest.java:48	High	3	Yuki MATSUBARA	d4e229

Figure 3: FindBugs warnings belonging to the CORECTNESS category.

When a student selects a warning message in Fig. 3, the corresponding code will be shown (Figure 4). Jenkins also shows the detailed explanation for the selected warning message.


Jenkins
保護されていない通信 | www.cs.is.saga-u.ac.jp/jenkins/view/2018年度グループ演習/job/Lily20…
Jenkins › 2018年度グループ演習 › Lily2018 › #5 › FindBugs Warnings › Category CORRECTNESS › PageDataTest.java

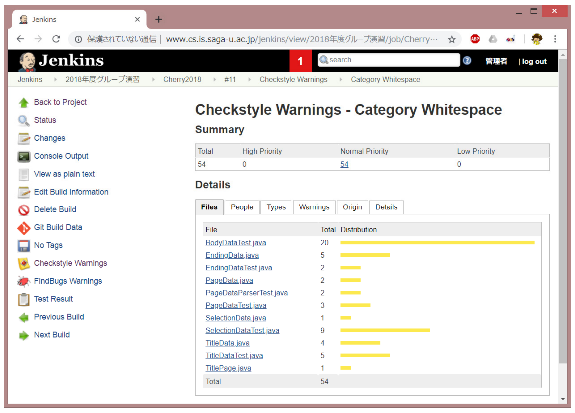
```
        // TODO 以下のテストを実装せよ
        // PageDataを生成する
PageData datap = new PageData();
        // 新規にTitleDataを作成し、setTitleData()で設定する
TitleData datat = new TitleData();
datap.setTitleData(datat);
        // getTitleData()の結果が生成したものと同じになることを確認する
assertEquals(datat.getText(),datap.getTitleData());
        // TODO 以下はテスト実装後削除すること
```

EC: Call to String.equals(sysdev.TitleData) in sysdev.PageDataTest.testTitleData01()

This method calls equals(Object) on two references of different class types and analysis suggests they will be to objects of different classes at runtime. Further, examination of the equals methods that would be invoked suggest that either this call will always return false, or else the equals method is not be symmetric (which is a property required by the contract for equals in class Object).

Figure 4: Code corresponding to a FindBugs warning.

Figure 5 represents messages of the coding standard checking. This time the warning messages are classified based on the detected file. It is also possible to browse the warning messages classified by the categories as in the case of Fig. 2.



Checkstyle Warnings - Category Whitespace

Summary

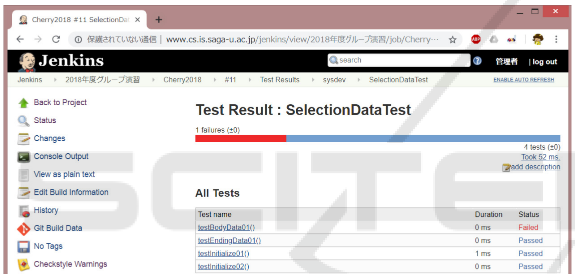
Total	High Priority	Normal Priority	Low Priority
54	0	54	0

Details

File	Total
BodyDataTest.java	20
EndingData.java	5
EndingDataTest.java	2
PageData.java	2
PageDataParserTest.java	2
PageDataTest.java	3
SelectionData.java	1
SelectionDataTest.java	9
TitleData.java	4
TitleDataTest.java	5
TitlePage.java	1
Total	54

Figure 5: Result of coding style checking.

Figure 6 represents a result of JUnit execution checking. The readers can find that one test failed while the remaining three tests succeeded. It is also possible to browse the corresponding source code by selecting the name of a test method.


Cherry2018 #11 SelectionDat
保護されていない通信 | www.cs.is.saga-u.ac.jp/jenkins/view/2018年度グループ演習/job/Cherry…
Jenkins
search
管理者
| log out
Jenkins › 2018年度グループ演習 › Cherry2018 › #11 › Test Results › sysdev › SelectionDataTest
ENABLE AUTO REFRESH
Back to Project
Status
Changes
Console Output
View as plain text
Edit Build Information
History
Git Build Data
No Tags
Checkstyle Warnings

Test Result : SelectionDataTest

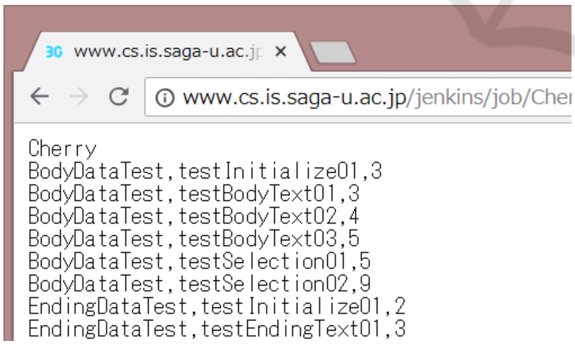
1 failures (±0)

4 tests (±0)
Took 52 ms.
add description

All Tests

Test name	Duration	Status
testBodyData01()	0 ms	Failed
testEndingData01()	0 ms	Passed
testInitialize01()	1 ms	Passed
testInitialize02()	0 ms	Passed

Figure 6: Result of JUnit execution checking.



```
Cherry
BodyDataTest,testInitialize01,3
BodyDataTest,testBodyText01,3
BodyDataTest,testBodyText02,4
BodyDataTest,testBodyText03,5
BodyDataTest,testSelection01,5
BodyDataTest,testSelection02,9
EndingDataTest,testInitialize01,2
EndingDataTest,testEndingText01,3
```

Figure 7: Example result of test case null implementation checking.

Figure 7 represents the result of test case null implementation checking. The result contains the test class name, test method name and the number of statements in the test method. In this case, for example, the method "testParse02" contains only one statement so that we can guess that the method is still at the initial state and is not implemented yet.

5 EFECT OF INTRODUCING ALECSS TO THE EXPERIMENT

Table 2 illustrates the size of each project code at the end of the first iteration (week 10). The number of files in each project is the same since file organization is determined by the teacher. Although implementation is different among the teams, the total code size is approximately 2000 lines. The readers can observe that we are using a reasonably large scale code for the experiment.

The number of test cases of the project is 27 as instructed from the teacher in the first iteration.

Table 2: Project size at week 10.

Project Team	# of Files	Total # of lines	Average # of lines per file
Cherry	19	2002	105.4
Dandelion	19	1997	105.1
Lily	19	1945	102.4
Peach	19	1962	103.3
Plum	19	1909	100.5
Rose	19	1703	89.6
Sunflower	19	1981	104.3
Violet	19	2019	106.3

Table 3: Project size at the end of the experiment.

Project Team	# of Files	Total # of lines	Average # of lines per file	# of Test Cases
Cherry	26	3130	120.4	44
Dandelion	25	2906	116.2	44
Lily	25	2712	108.5	37
Peach	26	3097	119.1	45
Plum	26	3159	121.5	52
Rose	25	3359	134.4	53
Sunflower	26	3227	124.1	43
Violet	25	3071	122.8	42

Table 3 represents the project size at the end of the experiment (week 15). Although file organization is determined by the teacher at the first iteration, several groups have extra classes. Implementation of the code is more different among the teams than the first iteration. The project size is significantly larger than Table 2. The number of test cases is distributed between 37 and 53 since the students had to add more than four originally test cases for each class added during the second iteration.

Table 4 represents the number of warnings generated by Checkstyle, FindBugs and the number of failed JUnit test cases at the time before review (week 9) and at the review (week 10). The number of test cases of all projects is 27 as instructed from the teacher in the first iteration. "N/A" means that the project failed to be compiled and the project code was not checked by the corresponding tool.

Table 4: The number of warnings and failed test cases of the first iteration.

Project Team	Checkstyle Warnings		FindBugs Warnings		Failed Test Cases	
	Week 9	Week 10	Week 9	Week 10	Week 9	Week 10
Cherry	141	69	1	1	7	0
Dandelion	N/A	66	N/A	1	N/A	0
Lily	95	74	5	7	2	4
Peach	100	81	1	1	4	3
Plum	97	76	3	7	8	1
Rose	78	64	2	1	7	0
Sunflower	73	63	7	2	0	0
Violet	105	N/A	2	N/A	10	N/A

The readers can observe that the number of warnings and the number of failed test cases are decreasing from week 9 to week 10 in most cases in the first iteration. This is the effect of peer review. Each team utilize ALECSS to detect and fix software bugs and potential problems by utilizing ALECSS. The number of Checkstyle warnings is still large at week 10. We consider that Checkstyle detects possible coding style errors regardless of their priorities and student teams do not have enough time to fix all the warnings with low priority.

However the number of warnings and failed test cases are increasing in some cases. One reason of this is that some project team did not finish the implementation at week 9 and continuously implement the code just before the peer review. Another reason is that some project team misunderstand our instruction and failed to correctly implement the test code at week 9.

Table 5 shows the number of warnings generated by Checkstyle, FindBugs and the number of failed JUnit test cases at the review (week 14) in the second iteration and at the end of the lecture (final). The readers can observe that the number of warnings and the number of failed test cases are decreasing from week 14 to final in all cases. This is the effect of peer review as same as the first iteration.

Table 5: The number of warnings and failed test cases of the second iteration.

Project Team	Checkstyle Warnings		FindBugs Warnings		Failed Test Cases	
	Week 14	Final	Week 14	Final	Week 14	Final
Cherry	169	178	33	33	11	3
Dandelion	134	130	19	13	15	6
Lily	120	156	16	29	2	3
Peach	186	188	25	20	7	8
Plum	233	194	24	24	11	10
Rose	101	104	45	38	9	5
Sunflower	141	137	43	42	4	3
Violet	126	126	33	29	7	3

Tables 6 and 7 represent the review score and the number of detected issues categorized by the type of issues at the peer review in the first and the second iterations respectively. The review score is calculated according to the rules which we have explained at the end of Section 3. The difference of the review score among the reviewer team is caused mainly by the utilization of ALECSS by the teams.

Table 6: Review score and the detected issues at the first iteration.

Reviewer Team	Review Score	# of Coding viola-tions	# of Algo-rithm issues	# of Bugs etc.	Reviewed Project
Sunflower	2	7	0	0	Cherry
Cherry	1	5	0	0	Dandelion
Rose	9	10	3	1	Lily
Dandelion	14	4	5	2	Peach
Peach	7	6	0	3	Plum
Violet	N/A	N/A	N/A	N/A	Rose
Lily	3	8	1	0	Sunflower
Plum	8	5	2	1	Violet

Table 7: Review score and the detected issues at the second iteration.

Reviewer Team	Review Score	# of Coding viola-tions	# of Algo-rithm issues	# of Bugs etc.	Reviewed Project
Sunflower	14	4	1	4	Cherry
Cherry	11	2	2	3	Peach
Rose	21	3	2	7	Lily
Dandelion	15	1	2	5	Rose
Peach	5	4	0	2	Dandelion
Violet	16	0	5	4	Plum
Lily	10	3	4	4	Violet
Plum	6	2	3	1	Sunflower

The results of Pearson correlation analysis among the numbers of warnings, the numbers of failed test cases and review score are illustrated in Table 8. Here, C10 and C14 are the number of Checkstyle warnings at week 10 and 14. F10 and F14 are the number of FindBugs warnings at week 10 and 14. And T10 and F14 are the number of failed test cases at week 10 and 14. Review1 and Review2 mean the review scores of iteration 1 and 2. In iteration 1 the numbers of Checkstyle warnings and the failed test cases are correlated with review results. In the second iteration, however, these are not correlated. It implies that the students could not detect the faults in the target codes in the second iteration because the code became more complicated, and the test cases were not enough because they were designed by the students although the test case specifications were supplied by the teacher in the first iteration.

Table 8: Pearson correlation analysis among results

Result Pair	correlation	t-value	Df	p-value
C10, Review1	0.903	4.204	4	0.014
F10, Review1	0.282	0.587	4	0.589
T10, Review1	0.849	3.207	4	0.033
C14, Review2	-0.455	-1.251	6	0.257
F14, Review2	0.0635	0.156	6	0.881
T14, Review2	-0.395	-1.053	6	0.333

6 STUDENT SURVEY

After the peer review at weeks 10 and 15, we conducted a student survey to evaluate ALECSS. The survey contains the following questions.

- Did you utilize ALECSS to check your project code at week 9? If no, why?
- Did you utilize ALECSS to find issues of another project at the peer review (week 10)? If no, why?
- How useful is ALECSS?
- Did you quickly obtain the feedback from ALECSS?
- Did you get a detailed result from ALECSS?
- Please provide comments to improve ALECSS.

57 of the 62 enrolled students at week 10 and all of them at week 15 answered the survey. We shall report the results of the survey in the succeeding subsections.

6.1 Utilization of ALECSS for Their Own Project

47 students (82.4%) at week 10 answered that they used ALECSS to check their own project. The ratio is quite high considering that they use ALECSS for the first time at week 9. The reasons of not using ALECSS at this week are as follows. Since some of the teams are still working on the implementation of the code, they could not have enough time to understand and utilize ALECSS for the checking of their own code.

The students also replied the following questions in five levels.

Q1. Was ALECSS useful to check your own project?

Q2. Did you quickly obtain the result from ALECSS?

Q3. Did you get the detailed result from ALECSS?

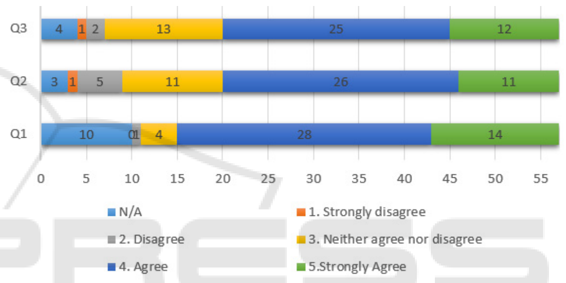


Figure 8: Evaluation of ALECSS at week 9.

As shown Figure 8, we obtained positive answers (5 or 4) from 89% of the students for usefulness, 65% for quick feedback and detailed checking. The number of negative answers (2 or 1) is quite few.

At week 15 after the second iteration, 55 students (89%) answered that they used ALECSS to check their own project. Figure 9 shows the results for the evaluation of ALECCS.

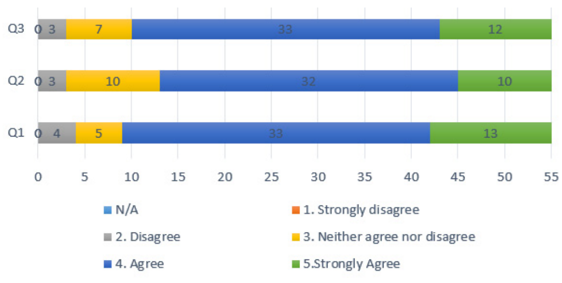


Figure 9: Evaluation of ALECSS at week 15.

We obtained positive answers (5 or 4) from 83.6% of the students for usefulness, 76.3% for quick feedback and 81.8% for detailed checking. The utilization of ALECSS increased during the second iteration.

6.2 Utilization of ALECSS to Review Other Project

52 students (91.2%) replied that they used ALECSS to find issues in another project at the peer review (week 10). Thus the percentage of the students is increased compared to the result at week 9. The reasons of not utilizing ALECSS at this week are as follows.

- Still did not understand ALECSS messages.
- Did not need ALECSS since Checkstyle reports possible coding style violations when I use Eclipse.
- Already knew that the reviewing code does not contain problems from the explanation provided by the teacher.

At week 15 after the second iteration of the experiment, 54 students (87%) replied that they used ALECSS to find issues in another project at the peer review.

They replied also if the system is useful in five levels at week 10 and week 15. Figure 10 shows the results.

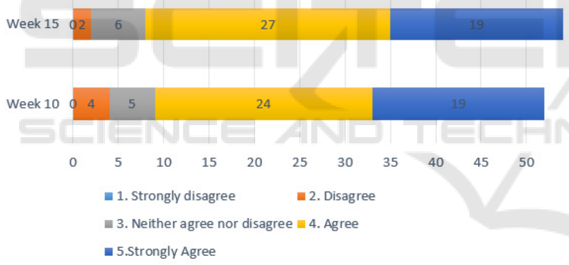


Figure 10: Evaluation of ALECSS at week 10 and week 15.

For week 10, we have Positive answers (5 or 4) are from 91% of the students and negative answers (2 or 1) from 9% of the students for usefulness of ALECSS. Then for week 15, we have Positive answers (5 or 4) are from 85% of the students and negative answers (2 or 1) from 4% of the students for usefulness of ALECSS.

We also collected the following comments about ALECSS. Although we obtained some negative comments, we are going to improve the system for further utilization.

- Excellent checking system
- Very useful since ALECSS provides detailed information about who did which task.
- Very useful since feedbacks are quickly provided.
- Hard to understand how to use ALECSS.
- I found some unexpected behaviour of ALECSS. Need to be fixed.

7 RELATED WORKS

Utilization of software tools for software engineering education has a long history (Douce, 2005; Ihantola, 2010; Ala-Mutka, 2007; Caiza, 2013). At that time, prevention of mindless resubmission of the student project was an important issue. Furthermore only few tools were available for free. On the other hand, ALECSS utilizes various open source software tools so that we are preparing to distribute ALECSS as open-source software for free. The concept of ALECSS is very similar to utilization of DevOps tools presented in Eddy's paper (Eddy 2017). Teachers can easily check progress and update history of the project by utilizing version management tool Git and continuous integration tool Jenkins. There are several education support systems for DevOps (Rong, 2017; Krusche, 2014). Unlike their work, we aim to support more general and flexible software engineering education using DevOps tools. ALECSS is designed to integrate various checking tools for the programming exercise so that we added original checking functions as we explained in Sections 2.1, 2.4, 2.5, 2.7 and 2.8.

Pape proposed a software tool STAGE for automatic grading of testing exercises (Pape, 2016). The tool utilizes CodeCover to measure several code coverage metrics in the context of white-box testing. It utilizes Moodle as a frontend to the students. However extension of the checking function is not addressed.

Nandigam et al. reported that student's understanding of various software engineering principles by utilizing various software tools (Nandigam, 2008). They also proposed to utilize software tools for iterative and incremental development, documenting software requirements, version control and source code management, coding standards compliance, design visualization, software testing, software metrics, etc. for undergraduate software engineering courses (Nandigam, 2014). Although the motivation of the research is similar to ours, they did not develop a system for automatic checking of the project code such as ALECSS.

Yu et al. also utilize free/open-source data and tools for upper-level software engineering class for two semesters, where instructor's experiences are assembled and analysed (Yu, 2014). They also reported that utilization of free/open-source data and tools facilitate understanding of the students and that their course can be kept up-to-date according to the advancement of the tools and development methods in industry. Along with the same experience as theirs, we developed ALECSS which can be utilized at other software engineering courses.

8 CONCLUDING REMARKS

Before developing ALECSS, the codes submitted by the students have been manually checked by the teachers. These checks are large burden for teachers and take time to return feedback to students. We often observe the similar situation at many exercise for programming education. ALECCS can automatically check the submitted codes so that students (as individual or team member) can check the results and correct their projects quickly. On the other hand, the teachers can monitor students' progress through Jenkins so that the burden of daily checking can be significantly decreased. Then the opportunity to guide students will be increased for the teachers.

For the future work, we are planning to integrate project management tool such as Redmine to maintain master schedule and integrate to ALECSS. Then the communication within each student team can be improved and the time management of the project will become easier. We also have a plan to distribute ALECSS by utilizing incremental deployment tool such as Chef or Docker. Then other educational institution can utilize ALECSS for their software engineering education.

ACKNOWLEDGEMENTS

This research is supported by JSPS Kakenhi Grant No. 17K01036. We also appreciate the students working on the development of ALECSS.

REFERENCES

Ala-Mutka, K. M., 2005. A survey of automated assessment approaches for programming assignments, *Computer Science Education Journal*, Vol. 15, Issue 2, pp. 83-102, DOI: 10.1080/08993400500150747

Allspaw, J., Hammond, P., 2009. *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr.*

Bass, L., Weber, I., Zhu, L., 2015. *DevOps: A Software Architect's Perspective, SEI Series in Software Engineering*, Addison-Wesley.

Caiza, J. C., Del Alamo, J. M., 2013. Programming assignments automatic grading: Review of tools and implementations, In *Proc. 7-th Technology, Education and Development* (INTEND 2013), pp. 5691-5700.

Douce, C., Livingstone, D., Orwell, J., 2005. Automatic test-based assessment of programming: A review, In *ACM Journal of Educational Resources in Computing, Vol. 5, No. 3*, Article 4, 113 pages.

Eddy, B. P. et al., 2017. CDEP: Continuous Delivery Educational Pipeline, In *Proc. South East Conference*, pp. 55-62, ACM.

Ihantola, P. et al., 2010. Review of recent systems for automatic assessment of programming assignments, In *Proc. 10-th Koli Calling International Conference on Computing Education Research*, pp. 86-93.

Koga, A., Feb. 2018. Test case checking functions for software engineer education support system ALECSS, As *Graduation Thesis*, Saga University. (in Japanese)

Krusche, S., Alperowitz, L., 2014. Introduction of continuous delivery in multi-customer project courses, *Proc. ICSE2014*, pp. 335-343, ACM.

Nandigam, J., Gudivada, V.N., Hamou-Lhadj, A., 2008. Learning software engineering principles using open source software, In *Proc. Frontiers in Education Conference*, FIE 4720643, pp. S3H18-S3H23.

Nandigam, J., Gudivada, V.N., 2014. Learning software industry practices with open source and free software tools, In *Open Source Technology: Concepts, Methodologies, Tools, and Applications 2-4*, pp. 997-1012.

Ohtsuki, M., Ohta, K., Kakeshita, T., 2016. Software engineer education support system ALECSS utilizing DevOps tools, In *Proc. 18-th International Conference on Information Integration and Web-based Applications & Services (iiWAS2016)*, pp. 209-213.

Pape, S., Flake, J., Beckmann, A., Jürjens, J. 2016. STAGE: A software tool for automatic grading of testing exercises: Case study paper, In *Proc. International Conference on Software Engineering*, pp. 491-500.

Rong, G. et al., 2017. DevOpsEnvy: An Education Support System for DevOps, In *Proc. CSEET2017*, pp. 37-46, IEEE.

Yu, L., Surma, D. R., Hakimzadeh, H., 2014. Incorporating free/open-source data and tools in software engineering education, In *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills*, pp. 431-441.