An Improved Java Programming Learning System Using Test-Driven Development Method

Nobuo Funabiki, Yuuki Fukuyama, Yukiko Matsushima, Toru Nakanishi, Kan Watanabe *

Abstract — **To enhance educational effects of Java programming by assisting self-studies of students and reducing teaching loads of teachers, we have proposed a *Web-based Java programming learning system using the test-driven development method*. In this system, a teacher should register Java programming assignments with statements, model source codes, and test codes using a Web browser. Then, a student can submit a test code and an answer source code for each assignment, where both codes are tested automatically by a testing tool called *Junit* at the server. Unfortunately, the current system cannot identify an incomplete test code that does not contain the complete test procedures if it has no grammatical error. In this paper, we introduce a code coverage measurement tool called *Cobertura* to detect such a test code by measuring the coverage rate when the submitted test code tests the model source code. We evaluate the effectiveness of our improved system through experiments with two simple assignments to 11 students who have studied Java.**

Keywords: Java education, Web system, test-driven development method, verification, code coverage

1 Introduction

Recently, with penetrations of the information and communication technology (ICT) into our societies, adverse affects of system failures due to software bugs have become intolerable. From this trend, the software test has been regarded as the last crucial process to avoid productions of software bugs in systems. Then, a *test-driven development (TDD) method* has been focused as an effective software developing method that can avoid bugs by writing and testing source codes at the same time [1]. In the TDD method, a test code should be written before writing a source code. A *test code* is a program code to verify the correctness of the outputs of the methods implemented in the *source code* to be tested.

Java is a useful and practical programming language that has been used in a lot of important systems including Web systems and embedded systems. Thus, the Java programming education in schools has been important so as to foster professional Java programmers. To enhance educational effects of the Java programming by assisting self-studies of students and reducing teaching loads of teachers, we have proposed a *Web-based Java programming learning system using the TDD method* [2]. In this Web application system, a teacher first should register assignments with their statements, model source codes, and test codes using a Web browser. Then, a student can submit a test code and an answer source code for each assignment. At the submission, these codes are tested automatically by using the code testing tool called *Junit* [3] at the Web server.

Unfortunately, the current system cannot identify an incomplete test code that does not contain the complete procedures of testing a source code if it has no grammatical error. In the TDD method, normally, the test code should generate an object of any class that is defined in the source code, and execute any path of the methods in the class for their tests. In our system, a test code from a student is tested through testing the model source code registered by a teacher. Even if this test code lacks some of the procedures for testing the methods in the model source code, it is passed. As a result, the correctness of the test code cannot be guaranteed in our current system.

In this paper, we additionally introduce a code coverage measurement tool called *Cobertura* [5] to detect an incomplete test code of a student by measuring the *coverage rate* of the model source code that is executed or covered by the test code. When the rate is not 1.0, the test code lacks testing procedures for some paths in the model source code. This means that the test code is incomplete, assuming the model source code is complete. Besides, this tool is used to find unnecessary descriptions in the source code of a student by measuring the coverage rate at the source code test by the teacher test code. When the rate is not 1.0, the source code contains unnecessary procedures that are not executed by the test code. We evaluate the effectiveness of the improved system through experiments to 11 students who have studied Java.

The rest of this paper is organized as follows: Section 2 introduces the TDD method. Section 3 reviews our Java programming learning system. Section 4 presents the improvement of our system. Section 5 discusses the evalu-

*Dept. of Electrical and Communication Engineering, Okayama University, 3-1-1 Tsushimanaka, Okayama, 700-8530, Japan Email: funabiki@cne.okayama-u.ac.jp

ation result. Section 6 concludes this paper with some future works.

2 Test-driven Development Method

In this section, we introduce the test-driven development method and its features.

2.1 Outline of TDD Method

In the TDD method, the test code should be written before the source code is implemented, so that it can verify whether the source code satisfies the required specifications during its development process. The basic cycle in the TDD method is as follows:

(1) to write the test code to test every specification,

(2) to write the source code, and

(3) to repeat modifications of the source code until it passes every test by the test code.

2.2 Junit

In our system, we adopt *Junit* as an open-source Java framework to support the TDD method. *Junit* can assist an automatic single test of a Java code unit or a *class*. Because *Junit* has been designed with the Java-user friendly style, its use including the test code programming is easy for Java programmers. In *Junit*, a test is performed by using a method whose name starts from "assert". This paper adopts the "assertEquals" method to compare the execution result of the source code with its expected value.

2.3 Test Code

The code test should be written using libraries in *Junit*. Here, by using the following *MyMath* class source code, we explain how to write a test code. *MyMath* class returns the summation of two integer arguments.

```
public class Math{
    public int plus(int a, int b){
        return( a + b );
    }
}
```

Then, the following test code can test the *plus* method in the *MyMath* class.

```
1: import static org.junit.Assert.*;
2: import org.junit.Test;
3: public class MathTest {
4:     @Test
5:     public void testPlus(){
6:        Math ma = new Math();
7:        int result = ma.plus(1, 4);
8:        asserEquals(5, result);
9:    }
10:}
```

The names in the test code should be related to those in the source code so that their correspondence becomes clear:

- The class name is given by the *test class name + Test*.
- The method name is given by the *test + test method name*.

The test code imports *Junit* packages containing test methods at lines 1 and 2, and declares *MathTest* at line 3. *@Test* at line 4 indicates that the succeeding method represents the test method. Then, it describes the test method.

The code test is performed as follows:

(1) to generate an instance for the *MyMath* class,

(2) to call the method in the instance in (1) using given arguments, and

(3) to compare the result with its expected value for the arguments in (2) using the *assertEquals* method.

2.4 Features in TDD Method

In the TDD method, the following features can be observed:

1. The test code can represent the specifications of the program, because it must describe any function to be tested in the program.
2. The test process for a source code becomes efficient, because each function can be tested individually.
3. The refactoring process of a source code becomes easy, because the modified code can be tested instantly.

Therefore, the study of the test code description is useful even for students, where the test code is equivalent to the program specification. Beside, students should experience the software test that has become important in companies.

3 Java Programming Learning System

In this section, we introduce the outline of the Java programming learning system in [2].

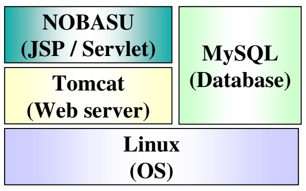


Figure 1: Server platform.

3.1 Server Platform

The Java programming learning system is implemented as a Web application. As the server platform, it adopts the operating system *Linux*, the Web server *Apache*, the application server *Tomcat*, and the database system *MySQL*, shown in Figure 1.

3.2 Two Learning Steps for Java Programming Self-studies

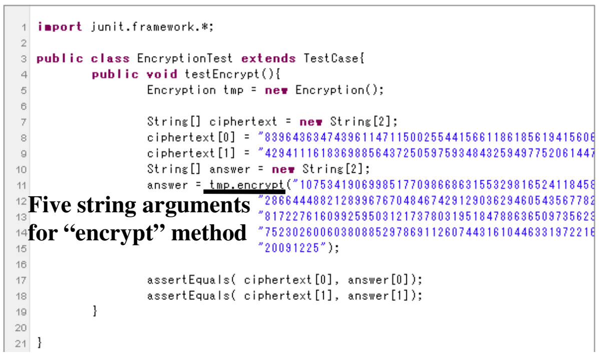
This system provides two learning steps for Java programming self-studies by students. One is a *Java code learning step* that allows students to write Java source codes by referring the test code made by a teacher. Another is a *TDD method learning step* that allows students to write both test code and source code, so that they can study the TDD method.

3.3 Java Code Learning Step

The Java code learning step consists of user functions for teachers and students. The *user functions for teachers* include the registration of new classes, the registration and management of assignments, and the verification of source codes submitted from students. To register a new assignment, a teacher needs to submit an assignment title, a problem statement, a model source code, and a test code, which are disclosed to the students except for the model source code after the registration. Note that the test code must test the model code correctly. Using the correspondence between a source code and a test code in Sect. 2.3, our system automatically generates a template for the test code from the model source code. Thus, a teacher only needs to specify figures for the arguments in each test method to complete the test code.

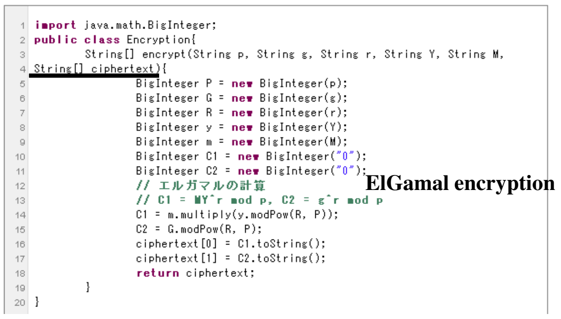
To evaluate the difficulty of assignments and the comprehension of students, it allows a teacher to view the number of submissions for code testing by each student. If a teacher finds that a lot of submissions have been tried by many students for an assignment, it can be considered too difficult for them and should be changed to an easier one. If a teacher finds a student who submitted codes many times whereas other students did so fewer times, the student should be cared specifically.

The *user functions to students* include the view of the



```
import junit.framework.*;

public class EncryptionTest extends TestCase{
        public void testEncrypt(){
                Encryption tmp = new Encryption();

                String[] ciphertext = new String[2];
                ciphertext[0] = "83964363474396114711500255441566118618561941560
                ciphertext[1] = "42941116183698856437250597593484325949775206144
                String[] answer = new String[2];
                answer = tmp.encrypt("107534190699851770986686315532981652411845
                "286644488212899676704846742912903629460543567782
                "817227616099259503121737803195184788636509735623
                "752302600603808852978691126074431610446331972216
                "20091225");

                assertEquals( ciphertext[0], answer[0]);
                assertEquals( ciphertext[1], answer[1]);
        }

}
```

Figure 2: Test code from teacher.



```
import java.math.BigInteger;
public class Encryption{
        String[] encrypt(String p, String g, String r, String Y, String M,
String[] ciphertext){
                BigInteger P = new BigInteger(p);
                BigInteger G = new BigInteger(g);
                BigInteger R = new BigInteger(r);
                BigInteger y = new BigInteger(Y);
                BigInteger m = new BigInteger(M);
                BigInteger C1 = new BigInteger("0");
                BigInteger C2 = new BigInteger("0");
                // エルガマルの計算
                // C1 = MY^r mod p, C2 = g^r mod p
                C1 = m.multiply(y.modPow(R, P));
                C2 = G.modPow(R, P);
                ciphertext[0] = C1.toString();
                ciphertext[1] = C2.toString();
                return ciphertext;
        }
}
```

Figure 3: Source code from student.

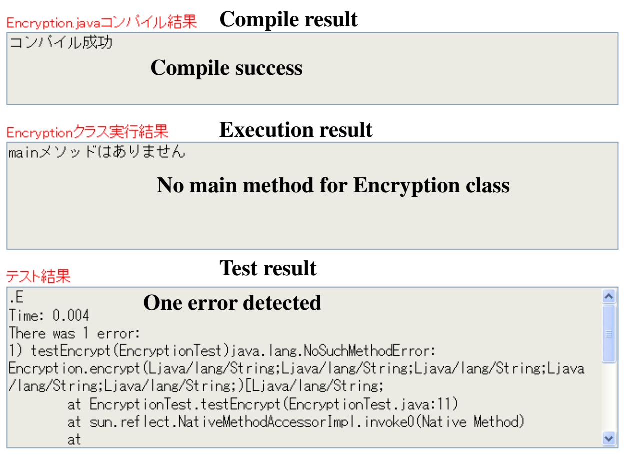
assignments and the submission of source codes for the assignments. A student should write a source code for an assignment by referring the problem statement and the test code, where he/she must use the class/method names, the types, and the argument setting specified in the test code. Our system implements a Web-based source code editor called *CodePress* [4] so that a student can write codes on a Web browser. The submitted source codes are stored in the database at the server so that they can view old ones.

Figures 2 and 3 show the test code and a source code from a student for the *ElGamal* encryption programming assignment. Figure 4 shows the test result, where one error is detected because the last argument of the "encrypt" method in the source code is different from the specification given in the test code.

3.4 TDD Method Learning Step

3.4.1 Name List

In this step, a *test code* from a student is verified through testing the model source code by the teacher, assuming that this model source code contains every necessary function correctly for the assignment. For this purpose, a teacher needs to disclose the class/method names, the



Encryption.javaコンパイル結果

```
コンパイル成功
```

Encryptionクラス実行結果

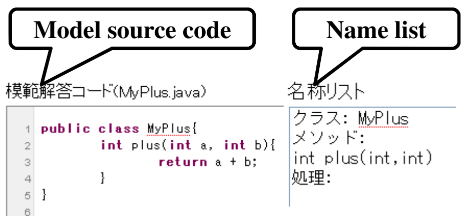
```
mainメソッドはありません
```

テスト結果

```
.E
Time: 0.004
There was 1 error:
1) testEncrypt(EncryptionTest)java.lang.NoSuchMethodError:
Encryption.encrypt(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava
/lang/String;Ljava/lang/String;)[Ljava/lang/String;
        at EncryptionTest.testEncrypt(EncryptionTest.java:11)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
```

Figure 4: Test result for source code.

types, and the argument setting in the model source code to students by using the *name list*. The name list contains every class and method name, the configuration of the arguments, and the return value type. To reduce the teacher load, our system automatically extracts the necessary information to generate the name list from the model source code, shown in Figure 5



模範解答コード(MyPlus.java)

```
public class MyPlus{
        int plus(int a, int b){
                return a + b;
        }
}
```

名称リスト

```
クラス: MyPlus
メソッド:
int plus(int,int)
処理:
```

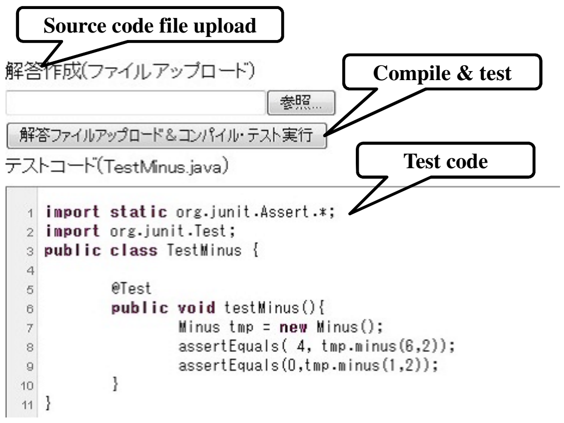
Figure 5: Model source code and name list.

3.4.2 Test Code Submission and Verification

A student should write a test code first by referring the problem statement and the name list from a teacher, and submit it to the server using the interface in Figure 6. Then, it is verified by testing the model source code of a teacher.

3.4.3 Source Code Submission and Verification

Then, a student writes a source code by referring the problem statement and his/her test code, and submits it. The source code is verified by both test codes from the teacher and the student. If both tests are passed, the test code and the source code can be regarded as correct ones. Otherwise, they are regarded as incorrect ones, and the student needs to fix the problems in them.


Source code file upload
Compile & test
Test code
解答作成(ファイルアップロード)
参照...
解答ファイルアップロード&コンパイル・テスト実行

テストコード(TestMinus.java)

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestMinus {

        @Test
        public void testMinus(){
                Minus tmp = new Minus();
                assertEquals( 4, tmp.minus(6,2));
                assertEquals(0,tmp.minus(1,2));
        }
}
```

Figure 6: Test code submission.

3.4.4 Problem in Test Code Verification

Unfortunately, this test code verification is not sufficient because it can be passed even if the test code lacks some necessary codes to test the corresponding functions in the source code. *Junit* tests the functions of the source code only if they are described in the test code, and cannot detect the lacks of testing even important functions. The improvement of this test code verification is actually a contribution of this paper, which will be discussed in the next section.

3.5 Secure Test Environment

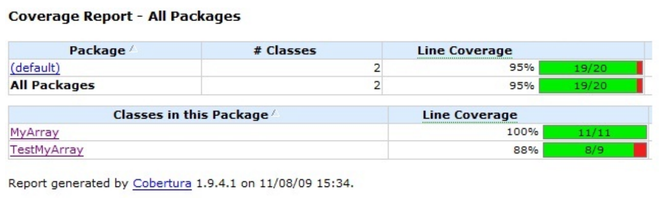
Here, we should note that source codes from students may contain defective commands such as an infinite loop and an illegal file access. To protect the server from them, two methods, namely a *code analysis* and an *execution time monitoring*, are adopted in our system. The code analysis analyzes the source code form a student to find whether it contains commands that may give adverse effects to the server. Specifically, if it contains a class for the file access such as "Buffered Write" and "PrintWriter", and a class for using external commands such as "Process" and Runtime", the test for this code is aborted and the test failure is returned to the student. The execution time monitoring executes the source code on a thread so that the execution time is observed from the main program. If the time exceeds a given limit or an exception is detected from the source code, the test is aborted and the failure is returned.

4 System Improvement by Code Coverage Measurement

In this section, we present the improvement of our system by introducing a code coverage measurement tool called *Cobertura*.

4.1 Outline of Cobertura

Cobertura can analyze the test coverage rate in the source code that is tested by the test code. The measurement report from *Cobertura* includes *line-rate* that represents the rate of the tested code lines in the source code, and *branch-rate* that represents the rate of the tested branches when branches exist. As these values approach to 1.0, the test coverage rate is improved. The result can be reported by an XML format or an HTML format so that it can be viewed graphically using a Web browser as shown in Figure 7.



Coverage Report - All Packages

Package	# Classes	Line Coverage	
(default)	2	95%	19/20
All Packages	2	95%	19/20

Classes in this Package	Line Coverage	
MyArray	100%	11/11
TestMyArray	88%	8/9

Report generated by Cobertura 1.9.4.1 on 11/08/09 15:34.

Figure 7: Coverage report by Corbertura.

4.2 Application of Cobertura

Cobertura is used when *Junit* tests a source code. In this subsection, we describe when this tool is used in our system.

4.2.1 Assignment Registration by Teacher

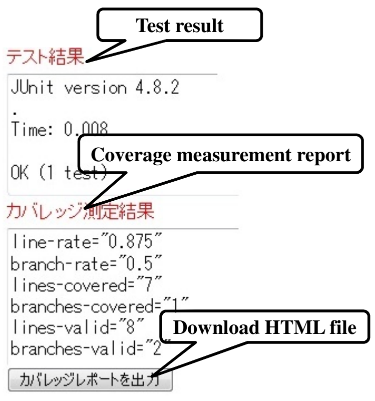
When a teacher registers a new assignment, *Cobertura* is used to verify whether the test code covers the tests of the model source code, so that errors in them can be avoided as best as possible.

4.2.2 Test Code Submission by Student

When a student submits a test code, *Cobertura* is used to verify whether the test code covers the tests of the model source code. The report is output with the test result as shown in Figure 8. If *line-rate* is 1.0, the test code covers the whole tests of the source code. If it is smaller than 1.0, the test code does not cover some tests. In this case, the student can download the coverage report of the HTML format. Here, we note that this report excludes the model source code by a teacher.

4.2.3 Source Code Submission by Student

When a student submits a source code, *Cobertura* is used to verify whether the source code contains unnecessary codes that are not tested by the test code from a teacher.



テスト結果

```
JUnit version 4.8.2
.
Time: 0.008

OK (1 test)
```

カバレッジ測定結果

```
line-rate="0.875"
branch-rate="0.5"
lines-covered="7"
branches-covered="1"
lines-valid="8"
branches-valid="2"
```

Figure 8: Test result of test code.

Table 1: Completions for two assignments.

Item	Assign. 1	Assign. 2
# of test code completions	-	8
# of source code completions	11	8

If the test code contains such unnecessary codes, they should be removed.

5 Evaluation

In order to evaluate the improvement of the Java learning system, we applied it to 11 students as experiments in our laboratory who have Java programming experiences.

5.1 Assignments and Test Results

For each of the two steps in our system, we prepared one assignment at the beginner level that is easy but requires multiple test methods in the test code. Specifically, *assignment 1* for the Java code learning step asks a source code for handling two-dimensional arrays, and *assignment 2* for the TDD method learning step asks a test code and a source code for extracting the maximum and minimum among the integer arrays. The completion results for them are shown in Table 1.

5.2 Questionnaire Results for Java Code Learning

As the questionnaire, we first asked them to reply the six questions in Tables 2 for Java code learning with five grades. Table 3 shows the results to them. The results to Q1 and Q2, where about one-third of the students replied 1 or 2, indicate that user interfaces of this system should be improved. The reason can be that the editor in the system does not work properly except on *FireFox*. The adoption of an editor that can work in any browser will be in our future works. The results to Q4 and Q5, where more than half of the students replied 2 or 3, indicate that the evaluation of the coverage measurement tool in writing a source code is not sufficient. The reason may be that the source code is short and can be written without modifications because the assignment is very easy. The evaluation using harder assignments will be necessary in future works. The results to Q6, where more than 80% of the students replied 4 or 5, indicate that this system helps students to understand the TDD method.

Table 2: Questions for questionnaire for Java code learning.

	Question
Q1	Do you think to use the system is easy ?
Q2	Do you think to read the test code is helpful in understanding the assignment specification ?
Q3	Do you understand the system outputs of the compiling and verification results easily ?
Q4	Do you think the coverage measurement tool in the system is helpful in learning Java codes ?
Q5	Do you think the graphical coverage report is useful ?
Q6	Do you understand the TDD method by using this system ?

Table 3: Questionnaire results for Java code learning.

		1	2	3	4	5	
Q1	no	1	3	0	4	3	yes
Q2	no	0	3	0	2	6	yes
Q3	no	0	1	2	5	3	yes
Q4	no	0	0	6	2	3	yes
Q5	no	0	1	6	3	1	yes
Q6	no	0	0	2	5	4	yes

5.3 Questionnaire Results for TDD Method Learning

Then, we asked them to reply the six questions in Tables 4 for TDD method learning with five grades. Table 5 shows the results to the questions. The results to Q4 and Q5, where more than half of the students replied 4 or 5, indicate that the coverage measurement tool is useful in writing the test code. Because no student in this experiment has experience in the TDD method, the coverage report can help them to write a correct test code of testing every required specification of the assignment. The results to Q3, where more than half of the students replied 3 or smaller grades, indicate that many students felt the response time of the system is long. Actually, in our measurement, the response time for one code submission becomes nearly 10 seconds. The reduction of this time by improving codes of the system and adopting multiple servers will be in our future studies.

Table 4: Questions for questionnaire for TDD learning.

	Question
Q1	Do you think to use the system is easy ?
Q2	Do you think to write the test code is helpful in understanding the assignment specification ?
Q3	Do you think the response time after the code submission is short ?
Q4	Do you think the coverage report is helpful in writing the test code ?
Q5	Do you think the graphical coverage report is useful ?
Q6	Do you understand the TDD method by using this system ?

Table 5: Questionnaire results for TDD method learning.

		1	2	3	4	5	
Q1	no	0	4	2	2	3	yes
Q2	no	1	0	4	4	2	yes
Q3	no(long)	1	2	3	1	4	yes(short)
Q4	no	0	0	4	2	5	yes
Q5	no	0	1	3	5	2	yes
Q6	no	0	0	3	6	2	yes

6 Conclusion

This paper presented an improvement of the Web-based Java programming learning system using the test-driven development method by introducing a code coverage measurement tool called *Cobertura*. It can detect an incomplete test code through measuring the coverage rate of testing the model source code by the test code. The evaluation results through experiments confirmed the effectiveness of the improved system. Improvements of user interfaces by using a code editor for multiple browsers, evaluations using harder assignments, and reductions of the system response time will be in our future studies.

References

[1] K. Beck, *Test-driven development: by example*, Addison-Wesley, 2002.

[2] N. Funabiki, T. Nakanishi, N. Amano, H. Kawano, Y. Fukuyama, and M. Isogai, "A software architecture and characteristic functions in learning management system "NOBASU"," Proc. SAINT 2010, pp. 109-112, July 2010.

[3] *Junit*, http://www.junit.org/.

[4] *CodePress*, http://sourceforge.net/projects/codepress/.

[5] Code coverage measurement tool *Cobertura*, http://cobertura.sourceforge.net/.