

# An Automated Assessment System for Analysis of Coding Convention Violations in Java Programming Assignments\*

HSI-MIN CHEN, WEI-HAN CHEN AND CHI-CHEN LEE  
*Department of Information Engineering and Computer Science*  
*Feng Chia University*  
*Taichung, 407 Taiwan*  
*E-mail: {hmchen; d0239867; gjl840311}@mail.fcu.edu.tw*

Coding conventions are a set of coding guidelines used by software developers to improve the readability of source code and increase software maintainability. Understanding coding conventions has become an indispensable discipline in software engineering; however, many university-level programming courses fail to prepare their students in this regard. In this study, we examined the distribution of coding convention violations in the assignments of programming courses where coding conventions are neglected. We then developed an automated system by which students can submit their programs and obtain immediate feedback on their coding assignments. Moreover, the system reduces the workload on instructors by providing insight into the quality of the code presented by students.

**Keywords:** code quality, coding convention, program assessment, code readability, software maintainability, software engineering

## 1. INTRODUCTION

The readability of source code is an important metric in characterizing the maintenance of software systems [1]. According to [2], maintenance can account for as much as 40% of the cost allotted for software development. Among software engineering techniques, coding convention is one of the major factors that have a significant impact on the readability of source code [3]. Coding conventions are a set of coding guidelines that software developers have to follow when they are writing code. Coding conventions are rules encompassing all concerns about improving code quality [4]. These rules are commonly categorized as follows: file structure, indentation, comments, white space, identifier naming, declarations, statements and practices specific to programming languages [5-10]. With coding conventions, readers can have a consistent look to source code and understand source code more quickly. In general, the first step for newcomers at most software companies involves familiarization with coding conventions.

Coding convention, a part of Clean Code [11], has been shown to improve productivity of software development and reduce program failures. Since most software systems are developed and maintained by a group of engineers, coding conventions make it quicker and easier for multiple team members to review and revise code. Despite its importance in the software industry, this issue is largely overlooked in programming courses at the university level. One impediment to the application of coding conventions in programming courses is the difficulty in reviewing programs to determine the degree

---

Received September 14, 2017; revised October 29, 2017; accepted December 19, 2017.  
Communicated by Yu-Chin Cheng.

\* This research was supported in part by Ministry of Science and Technology, Taiwan, under Grants No. 106-2221-E-035-002, 2017.

to which they adhere to defined coding conventions.

For example, the source code in Table 1 was written in accordance with the Google Java style [12], whereas the source code in Table 2 was written without regard for conventions (arbitrarily). Both of these source code segments have the same functionality, *i.e.*, converting temperature from Celsius to Fahrenheit. Furthermore, both of the code segments pass compilation and unit tests. Nonetheless, the source code in Table 1 is far easier to read than that listed in Table 2, which contains improper indentations, spacing, braces, identifier naming, and comments. Unfortunately, few instructors have the time to take coding conventions into account.

**Table 1. Example code following coding conventions.**

```

1  public class TemperatureConverter {
2
3      /**
4       * covert temperature from Celsius to Fahrehheit.
5       * @para Celsius Celsius temperature
6       * @return Fahrenheit temperature
7       */
8      public float covertToFahrenheit (float Celsius){
9          float fahrenheit = 32 + (9.0f/5.0f) * celsius;
10         return fahrenheit;
11     }
12 }
```

**Table 2. Example code with coding convention violations.**

```

1  public class TemperatureConverter {
2
3      public float covertToFahrenheit (float Celsius)
4  {
5      float fahrenheit =
6          32 + (9.0f/5.0f) * celsius;
7      return fahrenheit;
8  }}
```

In this research, we developed an automated assessment system for the analysis of code, especially with the ability to identify violations in coding conventions. The system was developed to examine Java programs submitted by students. Another goal of the system is to provide students with an iterative learning environment. Our aim was to enable students to submit coding assignments, obtain immediate feedback on the quality of their source code, and revise the code accordingly. The assessment system also furnishes a set of analysis reports for instructors and students alike, to help them better understand the status of assignments.

The main contributions of this research are as follows:

1. We developed a system that automatically assesses the code quality of assignments in

- Java programming courses, for use by students and instructors alike.
2. The system provides immediate feedback to facilitate the iterative development of programming skills.
  3. The system produces a variety of reports representing an overview of all assignments as well as the details of each program submission.

The remainder of this paper is organized as follows. Section 2 reviews related literature. Section 3 presents the distribution of coding convention violations in programming assignments in courses lacking instruction on coding conventions. The developed automatic assessment system is outlined in Section 4. The operational workflow of the system is demonstrated in Section 4. Conclusions are drawn in Section 5.

## 2. RELATED WORK

In recent years, a few schools have been aware of the importance of coding conventions in programming courses. Li *et al.* [13] spent three years to investigate whether students are willing to write code complying with coding standards. Their observation showed that students were reluctant to apply coding standards in writing code. To understand the cause, the authors made a questionnaire and distributed it to the students enrolled in three different programming courses. They found that more than 85% students considered coding standards was a significant topic. However, most students do not have time to apply coding standards in assignments. They proposed a set of teaching strategies to motivate students developing a habit of following coding standards. One of the strategies is introducing automatic tools to facilitate the compliance of coding standards when students learn to program.

In [14], the author proposed a code review process in a programming course in order to motivate students to apply coding standards and facilitate communications among students. The process includes choosing review materials, finding peers, and holding a review meeting, followed by submitting review reports. After the execution of a code review process, an interview with students, who participated in the process, was conducted to get their comments. The author found that applying coding review in programming courses can gain benefits, such as getting feedback from peers and forcing students to review code. However, it showed some drawbacks, including that students tend to give peers higher grades, and some students may cheat. Besides, it is difficult to apply the process in inspecting programs with large amount of code. Therefore, if there are automatic tools that can support code review, give correct feedback and assess code objectively, the issues of these drawbacks can be addressed.

Wang *et al.* [15] presented an improved peer code review process and analyzed the behavior of the participants. Different from traditional code review processes, its improved part is that students are allowed to continue sending code and getting review feedback to/from reviewers. When the revised code was complete, students can submit it to instructors for marks. After analyzing the participants' behavior, they found that the poor self-disciplined students did not consider the code quality when writing code as well as were bursting to send their code for review and waited for review comments. On the reviewer side, most reviewers were reluctant to review the code sent from poor pro-

gram writers. The authors said that the proposed process was not perfect due to some problems found. If automatic code review tools are available, they can facilitate the enforcement of process control, avoid conspiracy and reduce review time on reviewers.

In relation to the automatic detection of coding convention violations, a number of code analysis tools have been developed to examine violations of coding conventions and facilitate the improvement of code quality. CheckStyle [16] is a code analysis tool that can guide software developers to write Java code compliant with coding conventions. It provides more flexibility in configuring checking rules associated to different types of conventions. It also supports the generation of reports to present the types and locations of violations after programs are examined. In this research, CheckStyle is one of the tools integrated in our system to assess students' programming assignments.

PMD [17] is a source code analyzer able to help software developers discover potential flaws in programs. In terms of coding conventions, it can identify the violations, including braces, code size, naming and so forth. In addition, it takes programming problems into account, such as security flaw, duplicate code, unused code, coupling, design, optimization, exception, *etc.* The types of coding conventions covered by PMD are less comprehensive than those supported by CheckStyle. Customizing checking rules of PMD has a higher learning curve as compared to CheckStyle. PMD is usually served as a complement to CheckStyle to advance code quality.

SonarQube [18] is a web-based application that provides a set of features to source code analysis and a visualized dashboard to understand levels of code quality from various viewpoints, such as bugs, vulnerabilities, code smells, duplications, and code size. It can examine code quality continuously as long as programs are built and report distributions of coding problems over time. There are more than 25 programming languages supported by SonarQube thus far. SonarQube is developed as a platform that allows developers to integrate it with the tools of code inspection by implementing defined interfaces. Thus, it can leverage the existing tools, such as CheckStyle and PMD, to detect the violations of coding conventions and generate meaningful reports with friendly user interfaces to users.

As we used Git [19] to manage source code of assignments submitted by students, here are studies related to applying Git in programming courses. Kelleher [20] introduced Git technique and GitHub [21] Cloud service in in-class exercises and assignment homework. The author aimed to help students to be familiar with standard engineering tools and educated them to have programming ability demanded by software industry. The target students who learned Git are both second and third year CS undergraduate students. For second year students, the major Git topic they learned is cloning of repositories where starter code of assignments is provided. Third year students practiced more complicated operations of Git, such as creating own repositories, committing changes, creating branches and so forth. The author observed that using Git in classroom brings benefits including, secure submission, underwriting provenance, better source file organization, integrated issue tracking and exposure to standard industry practice.

Lawrance *et al.* [22] presented the benefits of using version control systems in classrooms and adopted Git as a version control tool taught in three CS courses, *i.e.* software engineering, compiler design and object-oriented programming, as well as CS1 courses for non-CS engineering majors. The authors described the process of setting up Git and leveraging GitHub in these courses to enable collaborative activities for students'

team projects. They learned that students appreciated using Git and continued to use Git outside of their courses.

Kertsz [23] presented a teaching model that adopted GitHub as a collaborative platform and used it in assignments of an operating system laboratory. Herein, GitHub features are employed in collaborative activities, including fork feature to distribute assignments to students, pull request feature to review code pushed by students and issue tracking feature to ask questions from students. With this model, the author can assess students' performances based on their collaborative logs, such as code commits, pull requests and raising issues, which can be retrieved by accessing the web APIs provided GitHub. A survey, made at the end of semester, pointed out most students preferred the collaborative platform. However, there exists a higher learning curve to understand the collaborative process of GitHub.

### 3. ANALYSIS OF CODING CONVENTION VIOLATIONS

In this section, we examine coding convention violations in assignments from two Java-related programming courses: Object-oriented Programming and Mobile Application Programming, offered at Feng Chia University, Taiwan. Neither of these courses guide students in the use of coding conventions. Our analysis revealed the types of coding convention violations most often made by students of programming courses.

#### 3.1 Checkstyle

Support tools, such as Checkstyle [16], PMD [17] and SonarQube [18], are helpful in identifying coding convention violations. These tools can facilitate the detection of coding convention violations in source code and provide recommendations on how to fix them. In this study, we opted for Checkstyle because it was designed specifically for the static analysis of Java code, it is highly configurable, and it is easily integrated with integrated development environments (IDEs) and build tools.

Checkstyle classifies coding conventions into the following 14 categories: Imports, Annotations, Block Checks, Class Design, Coding, Headers, JavaDoc Comments, Metrics, Modifiers, Naming Conventions, Regexp, Size Violations, Whitespace, and Miscellaneous. Users can customize their own check rules as checkers in each category. Checkstyle takes the checkers specified in a configuration file and the source code as inputs. It then parses the source code looking for violations and produces analysis results.

#### 3.2 Experiments and Results

We examined assignments from two Java-related programming courses in order to identify the types of coding convention violations commonly made by students. We employed Checkstyle in conjunction with Google Java Style as checkers in the analysis of source code. Google Java Style is widely used in Java software projects. In addition, most of its naming and formatting rules adhere to the original Java coding standard.

We began by analyzing the source code of assignments from a course on Object-oriented Programming. This included a total of eight assignments ranging from basic to advanced levels. Specifically, we examined student submissions for the first, fourth, and

eighth coding assignments. The first assignment involved writing a main method to serve as a program entry point and understanding the use of String APIs for the replacement of specific keywords for a given String object. The fourth assignment involved writing Java methods, implementing loop structures in methods, and invoking methods across objects. The final assignment involved programming a group of classes based on a given UML class diagram [24] and the implementation of Polymorphism principle [25].

As shown in Table 3, indentation violations were the most common problem. This can likely be attributed to students who get used to adopt tabs for indenting statements. The length of tabs is not fixed on all systems, which means that when code with tab indentations is opened on a different text editor, the presentation of the code may differ. It is for this reason that most coding standards recommend the use of spaces for indentation. Another problem was inconsistency in the indentations of statements belonging to the same level. Table 2 presents an example in which the indentation of statements on lines 5 and 7 should be the same. Maintaining consistency in the indentation of statements is viewed as important throughout the software industry.

**Table 3. Average number of violations in OOP assignments.**

Metrics	HW1	HW4	HW8
	Count		
Number of Classes	1	4	8
Lines of Code	8.5	91.2	112.6
Checker	Violations		
Indentation	5.5	38	50.4
Whitespace	1.4	27	38.1
Method_Def	0	2	5.2
Javadoc Comment	0.8	4.2	2.3
Identifier Naming	0.7	4	1
Brace	0	3.6	0.3

Whitespace was the second most common coding convention violation. Whitespace rule ensures that spaces are included before/after reserved symbols, such as commas, braces, parentheses, and operators. Table 2 presents an example where a whitespace should be placed between operator \* and variable Celsius on line 6. Violations in Method Def and Javadoc Comment were also common. The Method Def violation refers to a missing empty line between a method declaration and the statement above it. Javadoc Comment violations indicate the lack of a purpose-related comment corresponding to a public or protected method.

In the Java programming language, identifiers, including class name, method name, and variable name, have specific naming conventions. The class name should be a noun adhering to the upper camel case [26]. Method names should start with a verb. Variable names and method names should be in lower camel case [26]. Table 2 presents an example where class name, Temperature Converter, on line 1 does not adhere to the upper camel case, while variable name, Celsius, on line 3 is not in lower camel case. The last item in Table 3 refers to braces. The start brace should appear at the end of the same line as the declaration statement, and the end brace should start a line by itself. Table 2 illus-

trates an example where the start brace on line 4 should be placed at the end of line 3, and the end brace on line 8 should be moved to the next line with a correct indentation style.

We also examined team projects from a course on Mobile Application Programming. Students in this course formed teams to develop Android applications using Java programming language. By the end of the semester, 23 Android applications had been submitted. As in the previous course, the students were given no guidance with regard to coding conventions.

As shown in Table 4, these projects were riddled with two additional violations:

**Table 4. Average number of violations in Android Projects.**

Metrics	Count
Number of Classes	18
Lines of Code	3550
Checker	Violations
Indentation	1787
Whitespace	803
Method_Def	155
Javadoc Comment	54
Identifier Naming	248
Brace	203

Column Limit and Import Order. In Google Java Style, each line is limited to 100 characters, and import statements should follow the rule that import package names appear in ASCII [27] sort order. We can see that the number of violations in these team projects was far higher than that in the previous course. This resulted from that the instructor in the Object-oriented Programming course provided skeletons in starter code; however, no such starter code was provided in the Mobile Application Programming course. In addition, the size of assignment code in the Object-oriented Programming course is far smaller than that in the Mobile Application Programming course.

From these experiments, we discovered that "the definition of a completed assignment" for most students is that their code succeeds to execute and satisfies the functional requirements of assignments. The importance of coding conventions is neglected in most programming courses. If we can teach the concept of coding conventions in programming courses, which makes students' code evolve with clean and professional styles, it would be helpful to students to strengthen their competitiveness in software industry. Besides, since there are a large number of students enrolling and several homework assigned in a programming course, examining assignments for coding convention violations would impose a heavy burden on instructors and teaching assistants. These motivated us to develop an automated assessment system to promote programming skills by emphasizing good code quality, while helping to mitigate the work load on instructors.

#### 4. AUTOMATED ASSESSMENT SYSTEM

In this section, we outline the design rationale of the developed assessment system

as well as the features and key modules of the system.

#### 4.1 Design Rationale

Unlike existing programming assessment systems, the aim of the system is to help students learn programming as well as assist in evaluating code quality. Most assessment systems examine the source code of assignments, compare program outputs with test data, and provide graded assessment results. Our assessment system was designed as an iterative learning environment, where students submit programming assignments and obtain feedback pertaining to the quality of their code. This cycle is repeated until the assignment meets the necessary requirements or the assignment deadline comes due.

Fig. 1 presents the iterative learning process, which is initiated by an instructor who specifies a programming assignment. Once students are notified of the assignment, they can complete the assignment by writing code and submit it through the system. The system automatically analyzes assignments according to the assessment criteria defined by the instructor. Errors/failures detected during code analysis are flagged in feedback returned to the student. The students then revise the code accordingly and resubmit the assignment. The process does not end until the time of the deadline. Following each evaluation, a report is generated to advise the student of his/her submission status and code analysis results. Instructors can also see the progress made by the students according to the results of their submissions.

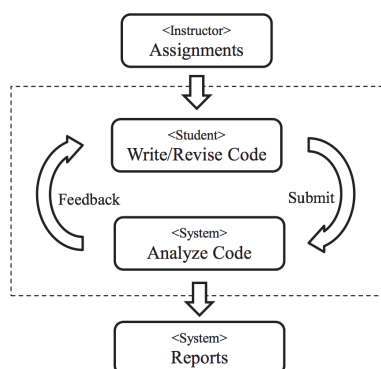


Fig. 1. The iterative learning process.

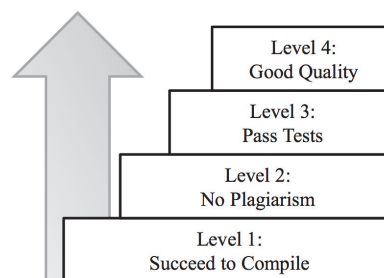


Fig. 2. Four levels of code quality, as determined by automated code analysis.

As shown in Fig. 2, we defined four levels of code quality. Level 1 code analysis involves checking whether the submitted programs succeed in compilation. Level 2 is a check for plagiarism, which can be disabled by the instructor for small programs. Level 3 involves evaluating the functionality of the program. Instructors can add test cases to determine whether the outputs of student programs match expectations. Level 4 involves analysis of code quality; *i.e.*, whether the students violated the coding conventions specified by instructors. In the future, we will also take into account potential bugs [28] and code smell [29].

The system halts code analysis and sends feedback to the student as soon as an error/failure is detected at any level. Students can use this to identify the cause of failures



and correct them in subsequent revisions. The logs of errors/failures and submissions are archived to serve as historical records for use in generating reports.

## **4.2 Features**

The developed web-based programming assessment system is called ProgEdu, the major features of which are detailed in the following.

### **4.2.1 Batch account registration**

Batch account registration makes it possible for instructors to batch-produce multiple accounts simply and easily. An enrollment file containing the identification number, name, and email of each student is first uploaded. The ProgEdu system then creates an account for each student automatically. Students can use the accounts to access ProgEdu as well as back-end third party services. Instructors can also utilize these accounts to trace the source code submitted by students.

### **4.2.2 Assignment management**

We furnished an assignment management feature to help instructors and students manage assignments more easily. ProgEdu provides a web-based user interface on which to set up assignments with descriptions, starter code, and coding convention rules. ProgEdu automatically creates corresponding workspaces for students to manage their programming assignments. When students sign in, workspaces are already created for them. This is where they can check out the latest code, make revisions, inspect the differences between two revisions, and trace their progress in the completion of each assignment.

### **4.2.3 Code analysis**

Since there are a large number of students enrolling in programming courses, we can imagine that reviewing correctness of code will take instructors and teaching assistants a lot of time. To reduce the review time, introducing automatic tools is necessary. We adopted a number of open-source tools to enable automated code analysis for the levels of code quality in order to lessen the load on instructors and teaching assistants.

### **4.2.4 Programming history**

Many universities have been developing learning management systems (LMSs) [30] to facilitate online learning. Moodle [31] is a widely adopted LMS working with a number of online course management services to enable the sharing of teaching materials, gauge learning performance, and produce statistical reports. This system also makes it possible for students to retrieve course materials and upload assignments from/to the Moodle system.

Despite Moodle provides many useful functions, it is essentially designed for general usage of course management. If students submit assignment programs through LMSs, what they deliver is a snapshot of source code. Instructors can only examine the

programs uploaded and check whether students submit them on time. However, in most assessment cases, understanding how students write code is more significant than what they submit finally. Existing LMSs lack a mechanism by which to trace the progress of students in learning to write code.

ProgEdu provides an iterative process that enables instructors and students alike to trace the progress of assignments students work on, and illustrate differences between revisions. This is highly valuable particularly for instructors seeking to assess the quality of the work completed by students as well as overall coding behavior.

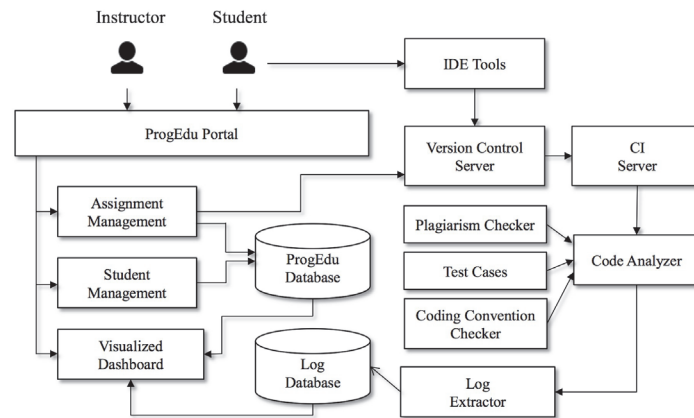


Fig. 3. Architecture of ProgEdu system.

#### 4.2.5 Timely reports

At present, instructors seeking statistical information on the learning performance of students must assess assignments, award grades, and provide comments on LMSs. Students are unable to obtain feedback from instructors until their completed assignments have been assessed.

ProgEdu has a reporting feature that gives students access to reports based on code analysis immediately upon submission of their programs. ProgEdu also provides instructors a variety of statistical charts indicating whether students have submitted their assignments on time and whether the quality of the submitted code meets expectations.

#### 4.3 System Architecture

Fig. 3 illustrates the architecture of the ProgEdu system, comprising a set of modules integrated with open-source tools. Details pertaining to each module are presented in the following. ProgEdu Portal serves as a front-end web-based user interface providing access to ProgEdu services for instructors and students. At the ProgEdu Portal, the instructor adopts the role of administrator, who has the authority to register student accounts, distribute assignments, and obtain statistical reports concerning the status of assignments. The ProgEdu Portal enables students to obtain links to assignment workspaces holding starter code, submit coding assignments, and check code analysis results.

The student management module enables the creation of student accounts on the ProgEdu and back-end Version Control Server, *i.e.* GitLab [32]. Unlike the cloud hosting services in [19], the server used for the management of student programs is located in our laboratory. This makes it possible to invoke Web APIs provided by GitLab for the management of student accounts, the creation of private repositories as programming workspaces, and access to logs related to each repository. The student management module makes it possible for instructors to set up student accounts individually or in batches in order to streamline workflow.

The assignment management module allows instructors to configure assignments by specifying descriptions, uploading starter code, and configuring code analysis via the ProgEdu Portal. The assignment module creates a corresponding Git [19] repository for the assignment of each student. The students can use their favorite IDE Tools, such as Eclipse [33], IntelliJ IDEA [34], NetBeans [35] and Android Studio [36], to submit assignments. We integrated the system with the open-source continuous integration (CI) server, Jenkins [37], to launch code analysis as soon as students submit assignments to their Git repositories.

The CI server periodically polls program submission events in GitLab. When students submit a new revision to GitLab, the CI server invokes the Code Analyzer to examine the revision. We extended the open source build tool, Maven [38], through the development of three plugins: plagiarism, unit test, and coding convention. Our Log Extractor module retrieves analysis results and logs from the Code Analyzer. The bundle of retrieved information includes a student identifier, a timestamp when the student submitted the code, a comment describing the submission, the names of files revised in that submission, and code analysis results including data pertaining to failures. The Log Extractor module stores this data in a Log Database (an open-source document-based database called Elasticsearch [39]) to enable the processing of data for further analysis and visualization. Unlike relational databases, Elasticsearch is easy to scale up, supports schema-free data structures, and has outstanding search and analysis performance.

The Visualized Dashboard module provides users with a set of visualization reports, including tables and charts. It interacts with the Log Database to retrieve relevant data and generate corresponding reports based on user demands. These reports make it possible for instructors to visually assess assignments using a variety of indices on the dashboard. Students can obtain feedback from the reports as well as the status of each assignment. They can also obtain rich information regarding of the types of failure they encountered in each assignment as well as methods to correct them.

## 5. SYSTEM DEMONSTRATION

Fig. 4 presents the operational scenario of assigning homework projects, completing assignments, and obtaining code analysis results. Instructors sign into the system and register student accounts. They then upload an entire enrollment list to enable the automatic creation of student accounts. At the end of each course unit, the instructor then assigns an assignment.

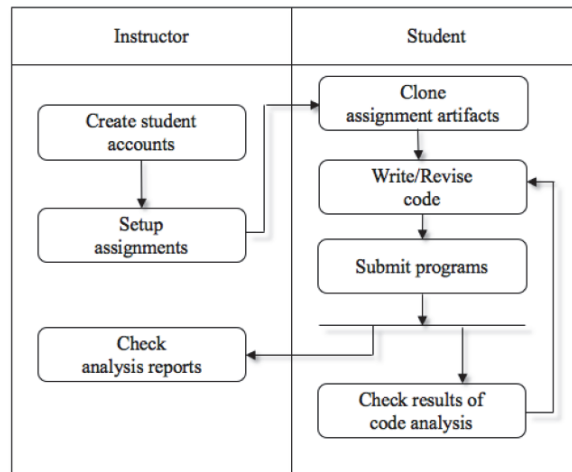


Fig. 4. Operational scenario of OpenEdu.

After that, the students receive an email notification describing the assignment as well as links to assignment repositories on the GitLab server. Students can employ their favorite IDE tools equipped with Git client functionality to clone relevant artifacts from remote repositories to their local programming workspaces. The students then submit the source code from completed assignments to their repositories using Git push command. Git is the most popular version control system in use today. Through ProgEdu, students are able to learn the management of source code with version control systems, especially with Git.

As shown in Fig. 5, each student has his/her own Web page on which to obtain an overview of assignments from the ProgEdu Portal. The overview contains a table indicating the number of projects assigned by the instructor thus far, the number of submissions related to each assignment, and the final code analysis results marked in various colors. To see the details of a particular assignment, students simply press one of the links on the left side of the page to display a detailed report, as shown in Fig. 6. The location of the Git repository associated with this assignment is presented at the top of the page. The current code analysis results are presented below the Git repository link. The number is the submission count, and the color indicates the level of code quality, ranging from poor (gray) to good (green). The programming history, containing the submission date, comments and the code analysis result of each submission, is listed next to the code

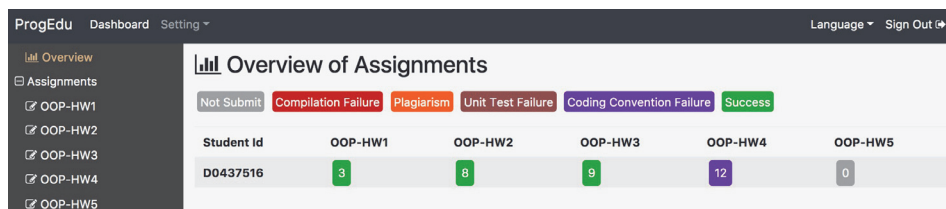


Fig. 5. The overview of assignments for students.

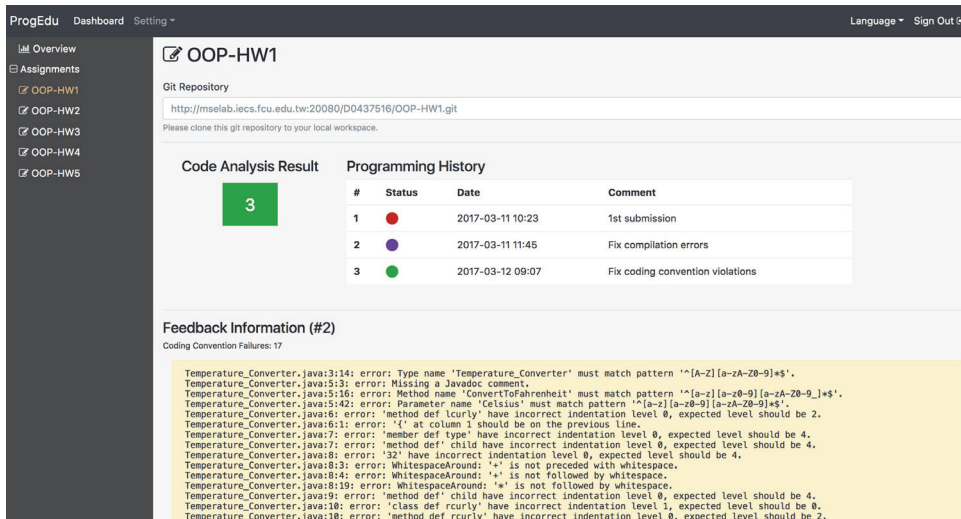


Fig. 6. Assignment report for students.

analysis results. After students click on one of submission records in the programming history, feedback information for that submission is presented at the bottom of the page. Information displayed in this area lists the causes of errors/failures associated with each submission.

ProgEdu also provides instructors with a set of reports by which to assess assignments using a variety of indices to identify the types of difficulties students face while learning to write code. Fig. 7 presents a table illustrating the state of each assignment submitted by each student. The colors marked in the rounded rectangles indicate the level of code quality.

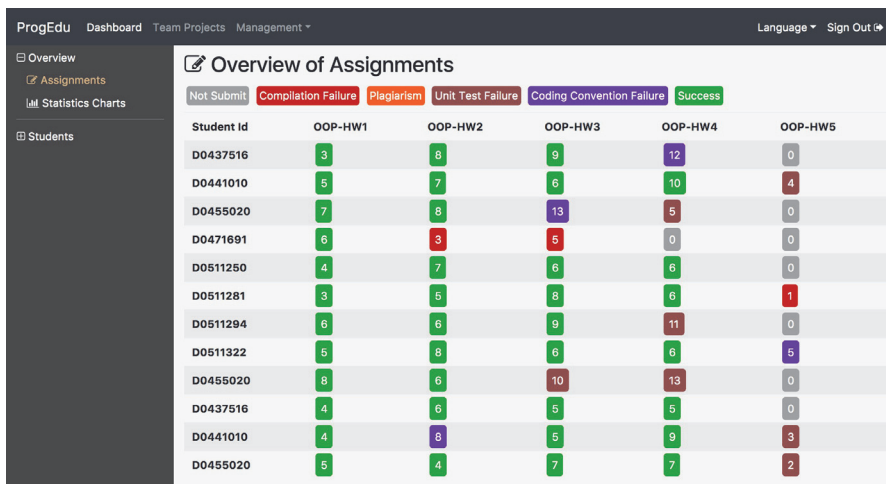
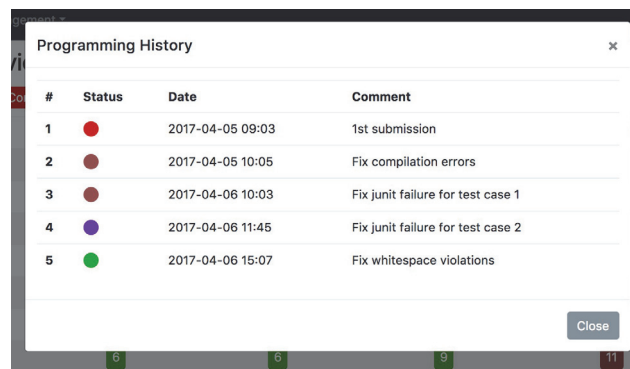


Fig. 7. Overview of assignments for instructors.



#	Status	Date	Comment
1	●	2017-04-05 09:03	1st submission
2	●	2017-04-05 10:05	Fix compilation errors
3	●	2017-04-06 10:03	Fix junit failure for test case 1
4	●	2017-04-06 11:45	Fix junit failure for test case 2
5	●	2017-04-06 15:07	Fix whitespace violations

Fig. 8. Programming history of one assignment.

As shown in Fig. 8, when instructors click on the number of code submissions shown in the rounded rectangles, ProgEdu Portal opens a new window with the programming history of the assignment a student has been working on. The programming history contains a series of submission records. They can also click on the status circle of that submission to obtain the corresponding feedback information, as shown at the bottom of the page in Fig. 6. This gives instructors a broad understanding for all the programs submitted by all students as well as the code analysis details of each submission.

ProgEdu also furnishes a variety of charts to assist instructors in analyzing assignment statuses. Currently, most LMSs only provide instructors with text-based statistic information about turn-in status of each assignment. ProgEdu provides a chart presenting the distribution of program submission status with respect to each assignment, as shown in Fig. 9. This chart shows instructors the number of students who handed in their assignments and whether the submissions were on time. With this chart, instructors can understand the trend of assignment submissions. If the number of on-time submission gradually decreases over time, it would indicate that students encounter difficulties in learning programming. Instructors should slow down the pace of teaching.

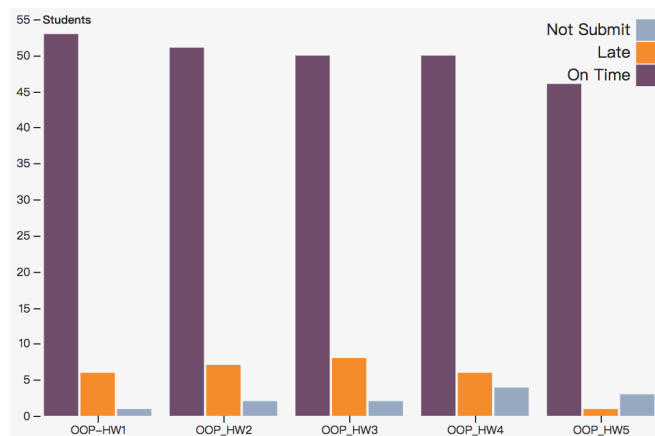


Fig. 9. Distribution of program submission status.

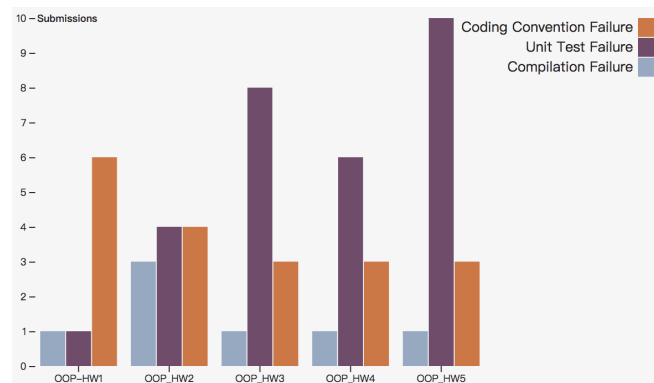


Fig. 10. Distribution of code analysis failures.

Most programming assessment systems are capable of determining whether submitted programs meet the functional requirement of assignments, but not able to further identify the types of programming failures. ProgEdu provides a chart showing instructors the average number of failures for different types made by students in relation to each assignment, as depicted in Fig. 10. With this chart, instructors can investigate which kinds of programming failures that students made commonly in each assignment and then can spend time to teach students how to resolve these failures.

In terms of coding conventions, ProgEdu provides a detailed view to show the average number of coding convention failures detected in each assignment, as depicted in Fig. 11. With this chart, instructors can understand which types of coding conventions that students are not familiar with, even though immediate feedback is replied. In response to a higher number of coding convention failures, instructors can consider to improve the recommendation of failure fixes in feedback information or seek for tools that students can employ to correct failures when they are writing code.

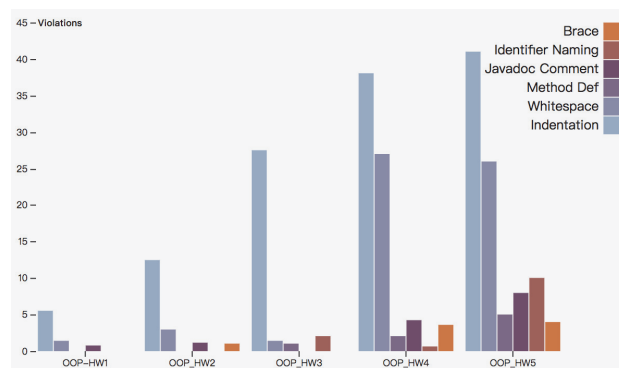


Fig. 11. Distribution of coding convention failures.

These charts have three benefits: (1) reduced workload in the review of assignments, enabling teachers to concentrate on teaching tasks that cannot be performed by machines; (2) information on the progress of writing code and the types of difficulties faced by

students; (3) analysis from various perspectives, including submission status, code analysis results, programming history, and code quality.

In order to examine the effect of the developed system, we applied it in the course of Object-oriented Programming in the first semester of 2017. There were 60 students enrolled in this course, and seven assignments were assigned thus far. Fig. 12 depicts the analysis result in which each bar presents the number of submissions over the total students. Because the first three assignments are basic and simple, the function of coding convention detection was not enabled. We can see that the average number of submissions is around two. There were no coding convention failures detected in these three assignments.

After the third assignment, the instructor taught the concept of coding convention and asked students to write code in accordance with the Google Java style. From the analysis result in Fig. 12, it shows additional three iterations spent by students to fix coding convention failures with respect to homework 4. This resulted from that they were not familiar with the rules of Google Java Style. In relation to the homework from 5 to 7, the number of submissions for the fixes of coding convention failures decreased gradually. This indicates that students are able to submit programs that meets the required coding conventions. This empirical study presents the actual improvement of students' program skills in terms of code quality by employing the iterative learning mechanism.

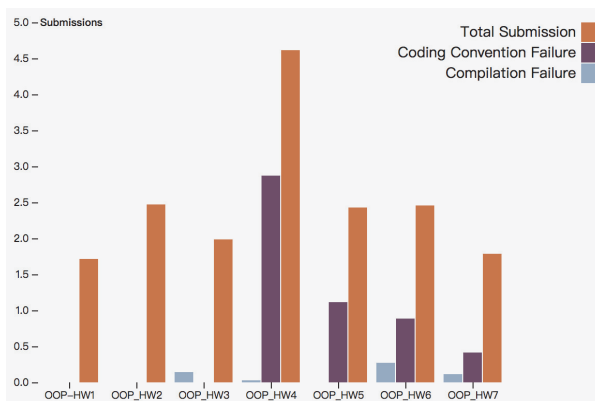


Fig. 12. Effect of automatic coding convention detection.

## 6. CONCLUDING REMARKS

In this paper, an automatic assessment system to analyze code quality for Java programming courses is presented. Different from other assessment systems that grades students' programs by comparing the output data with expected ones, the system can facilitate students to continue revising their programs until the assignment requirements including succeeding to compilation, no plagiarism, passing unit test and no violations of coding conventions, are fulfilled. Through the iterative process, students can learn programming from the feedback related to various levels of code analysis. Thus, the system



can assist students in advancing programming skills with good code quality. In addition, with the support of the system, instructors can reduce their time and resources on reviewing source code and assess students programs from various dimensions of analysis results.

Currently, the system is developed as a basic infrastructure to support the assessment of programming assignments. In the future, we will extend the system to satisfy the following requirements:

1. The system can facilitate students to develop team projects and help instructors analyze programming behavior from the data of programming histories. We plan to mine meaningful information, for example, the contribution of each team member, the patterns of plagiarism, the association between submission count and final grade, *etc.*, from the data collected by the system.
2. The system can support complex code analysis such as the detections of potential bugs and bad smell so that the quality of students programs can get further improved. Besides, the system can support code analysis for more programming languages, such as C, C++ and Python.
3. Nowadays, there are many online programming courses offered by MOOCs (Massive Open Online Courses) [40] platforms. However, it lacks supporting tools that allow learners to practice programming and analyze their code quality automatically. Thus, we plan to leverage container technology [41] to turn the system into a supporting tool so that it can be easily integrated with MOOCs platforms.

## REFERENCES

1. R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, Vol. 36, 2009, pp. 546-558.
2. R. L. Glass, *Facts and Fallacies of Software Engineering*, Addison-Wesley, Boston, 2002.
3. Y. Tashtoush, Z. Odat, I. Alsmadi, and M. Yatim, "Impact of programming features on code readability," *International Journal of Software Engineering and Its Applications*, Vol. 7, 2013, pp. 441-458.
4. M. Smit, B. Gergel, H. J. Hoover, and E. Stroulia, "Code convention adherence in evolving software," in *Proceedings of the 27th IEEE International Conference on Software Maintenance*, 2011, pp. 504-507.
5. Google, "Google's coding standards for source code in the Java™ Programming Language," *Google Style Guides*, <https://github.com/google/styleguide>, 2017.
6. Facebook, Coding Standards, <https://github.com/facebook/jcommon/wiki/Coding-Standards>, 2016.
7. K. Cwalina and B. Abrams, *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*, Addison-Wesley, Boston, 2005.
8. A. Reddy, *Java Coding Style Guide*, Sun Microsystems, 2000.
9. H. Sutter and A. Alexandrescu, *C++ Coding Standards: 101 Rules, Guidelines, and Best Practices*, Addison-Wesley, Boston, 2004.
10. F. Lepied, *Quality Python Development*, 2012.

11. R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Prentice Hall, 2008.
12. Google, "Google Java style guide," <https://google.github.io/styleguide/javaguide.html>, 2017.
13. X. Li and C. Prasad, "Effectively teaching coding standards in programming," in *Proceedings of the 6th Conference on Information Technology Education*, 2005, pp. 239-244.
14. X. Li, "Using peer review to assess coding standards a case study," in *Proceedings of the 36th Annual Conference on Frontiers in Education*, 2006, pp. 9-14.
15. Y. Wang, L. Yijun, M. Collins, and P. Liu, "Process improvement of peer code review and behavior analysis of its participants," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, 2008, pp. 107-111.
16. Checkstyle, <http://checkstyle.sourceforge.net/>, 2017.
17. PMD, <https://pmd.github.io/>, 2017.
18. SonarQube, <https://www.sonarqube.org/>, 2017.
19. D. Spinellis, "Git," *IEEE Software*, Vol. 29, 2012, pp. 100-101.
20. J. Kelleher, "Employing git in the classroom," in *Proceedings of World Congress on Computer Applications and Information Systems*, 2014, pp. 1-4.
21. A. Pipinellis, *GitHub Essentials*, Packt Publishing, UK, 2015.
22. J. Lawrance, S. Jung, and C. Wiseman, "Git on the cloud in the classroom," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 2013, pp. 639-644.
23. C. Z. Kertesz, "Using github in the classroom a collaborative learning experience," in *Proceedings of the 21st IEEE International Symposium for Design and Technology in Electronic Packaging*, 2015, pp. 381-386.
24. J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual*, Pearson Higher Education, US, 2004.
25. B. Eckel, *Thinking in Java*, Prentice Hall, NJ, 2006.
26. F. P. Miller, A. F. Vandome, and J. McBrewhster, CamelCase: Compound (Linguistics), Whitespace (Computer Science), Capitalization, Patti LaBelle, Visual Basic, MacGyver, iPod, Chemical Formula, Naming ... Programming Language, Marketing. Alpha Press, 2009.
27. C. E. MacKenzie, *Coded Character Sets: History and Development*, Addison Wesley, Boston, 1979.
28. N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh, "Using static analysis to find bugs," *IEEE Software*, Vol. 25, 2008, pp. 22-29.
29. M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, Boston, 1999.
30. Y. Kats and Y. Kats, *Learning Management Systems and Instructional Design: Best Practices in Online Education*, IGI Global, PA, 2013.
31. S. S. Nash and M. Moore, *Moodle Course Design Best Practices*, Packt Publishing, UK, 2014.
32. J. van Baarsene, *GitLab Cookbook*, Packt Publishing, UK, 2014.
33. R. Kulkarni, *Java EE Development with Eclipse*, Packt Publishing, UK, 2015.
34. J. Krochmalski, *IntelliJ IDEA Essentials*, Packt Publishing, UK, 2014.
35. G. Wielenga, *Beginning NetBeans IDE: For Java Developers*, Apress, NY, 2015.

36. C. Craig and A. Gerber, *Learn Android Studio: Build Android Apps Quickly and Effectively*, Apress, NY, 2015.
37. N. Pathania, *Learning Continuous Integration with Jenkins*, Packt Publishing, UK, 2016.
38. B. Varanasi and S. Belida, *Introducing Maven*, Apress, NY, 2014.
39. C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*, O'Reilly Media, CA, 2015.
40. J. Kay, P. Reimann, E. Diebold, and B. Kummerfeld, "Moocs: So many learners, so much potential ..." *IEEE Intelligent Systems*, Vol. 28, 2013, pp. 70-77.
41. D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, Vol. 1, 2014, pp. 81-84.



**Hsi-Min Chen (陳錫民)** received the B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Central University, Taiwan, in 2000 and 2010, respectively. He is currently an Assistant Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, object-oriented technology, service computing, and distributed computing.



**Wei-Han Chen (陳薇涵)** received the B.S. degree in the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. She is currently a master student in the same department at Feng Chia University. Her research interests include software engineering, mobile application technology, and education technology.



**Chi-Chen Lee (李霽丞)** received the B.S. degree in the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. He is currently a master student in the same department at Feng Chia University. His research interests include software engineering, mobile application technology, and education technology.