https://doi.org/10.48009/3_iis_2016_165-175

Issues in Information Systems
Volume 17, Issue III, pp. 165-175, 2016

ENHANCING THE LEARNING PROCESS IN PROGRAMMING COURSES THROUGH AN AUTOMATED FEEDBACK AND ASSIGNMENT MANAGEMENT SYSTEM

Xue Bai, Virginia State University, xbai@vsu.edu
Ade Ola, Virginia State University, aold@vsu.edu
Somasheker Akkaladevi, Virginia State University, saakaladevi@vsu.edu
Yingjin Cui, The Governor's School @ Innovation Park, ycui2@gmu.edu

ABSTRACT

One of the best methods of learning in computer programming courses depends on practical exercises accompanied by a good feedback procedure. Feedback on programming assignments has a significant impact on learning; it has been described as the most powerful single motivator that enhances achievement. However, the process of preparing, collecting and grading programming assignments manually takes time. Particularly, manual grading of programming assignments to provide effective feedback is a tedious and time consuming task. As a result, the number of assigned programming lab work is often fewer than is ideal. Because providing effective feedback on programming assignments is so cumbersome, it is difficult to provide feedback in a timely manner. These problems can reduce the effectiveness of the programming learning process. This paper describes an online assignment management system and a hierarchical auto-grading structure for automatic evaluation of programming assignments, which allows instructors to distribute homework online and to offer fast and effective feedback. The automatic grading and feedback generation system has been used in actual courses at our university. Collected data confirm that instructors assigned more homework and provide quicker feedback when using the system than without.

Keywords: Auto-grading, Feedback, Assignment Management

INTRODUCTION

One of the primary functions of an academic unit that teaches computer programming courses is to imbue its students with the ability to program. It has been shown that the most effective methods of learning in programming courses depends on practical exercises (Cheanga, Kurniaa, Limb, and Oonc, 2003; Cheang, Kurnia, Lim, Wee-Chong, 2003). Lab work, homework and project assignments provide ways for students to translate their theoretical knowledge into designing computer programs and to practice their programming skills. The main path to improve programming skills has been the increment of solved programming exercises, and this has to be accompanied with a good feedback. However, preparing, collecting and grading programming assignments manually take time. Particularly, manual grading of programming assignments to provide effective feedback is a tedious and time consuming task. Thus, feedback often comes days, weeks, or even months after the assignments are collected. But for feedback to be effective, it has to be provided in a timely manner. The sooner students receive feedback, the better chance that they would read, understand, and use the feedback to improve their programming skills. But the reality is that programming assignments are still managed manually in most classrooms. Programming assignments are distributed, typically, through printouts or via Internet (perhaps via a Learning Management System (LMS) or email). Upon completion, students submit their work via email or upload them to an LMS system. The instructor then downloads submitted program files for compilation and testing, then provide feedback based on testing results. This manual process is cumbersome and time-consuming. The process works with small number of students; however, when the number of student increases, productivity decreases and human related errors tend to increase on the instructor side. The manual process and the associated problems reduce the number of homework exercises that can be assigned and the effectiveness of feedback. Also, when feedback is not provided promptly, students are not able to determine whether they are on the right track when devising solutions to a given homework problem. It is commonly reported that students do not read the instructor's feedback comments (Duncan, 2007), especially when the feedback does not come in a timely manner. The literature suggests that part of the problem is that teachers (and

students) see feedback in isolation from other aspects of the teaching and learning process, and students consider feedback to be primarily a teacher-owned endeavor (Taras, 2003). Therefore, automatically providing feedback on programming assignments in a timely manner is becoming more of a need than an option (Caiza and Del Alamo, 2012). The motivation for designing the system hereby presented is to provide a solution to the problems associated with providing effective feedback and managing programming assignments and to offer tools that will enhance the learning process.

There are a few automated grading systems in the literature or on the market. Moodle, one of the most prominent LMS, provides a plug-in called Virtual Programming Lab (VPL), which requires a dedicated execution server. The server runs supplied test scripts on programs submitted by the students. However, Moodle lacks the tools for compiling and executing programming assignments (J.M. del Alamo, A. Alonso, M. Cortés, 2012). Besides, the system requires a significant amount of work before an assignment can be released. The instructor must spend time preparing an assignment, writing a solution program, generating scripts that will evaluate the program, testing the complete setup, and incorporating all these components as a new VPL Activity in Moodle (Thiébaut, 2015). Codebat, Betterprogrammer, and Practice-it are commercial tools which provide users with instantaneous feedback. Both CodingBat and Practice-it permit the user to re-attempt problems until the correct solution is achieved. Another good feature of the CodingBat and practice-it is the authoring facility where users can contribute their own programming problems to the existing set. However, these systems can handle only small pieces of code; they are not suitable for managing large-scale programming assignments.

Beierle and Widera (2003) presented an approach to the automatic revision of homework assignments in programming language courses. Their work provided an abstract framework, AT(x) (analyze-and-test for a language x) system, which analyzes programs and generates comments automatically (Dutton, 2001). There are other categories of tools for conducting programming examinations online. Roberts and Verbyla (2003) proposed one such tool that allows Java programming exams to be conducted securely online. However, none of the systems reviewed effectively addresses all the issues relating to management and grading of programming assignments that are discussed in this paper.

AUTOMATED ASSIGNMENT MANAGEMENT AND FEEDBACK SYSTEM

We have developed a web-based assignment management system to facilitate learning in computer programming courses. The system provides a stable software development environment for students' programming assignments and projects. It uses a remote application server to compile and run students' solutions to programming assignments. The system is a pure web-based application that is accessible from any type of browser. The programming environment that is configured and installed on the remote server includes java runtime environment, compiler, databases, externals files, services, and external APIs. The configuration sets up an environment where all students have access to identical software development environment from any location where there is an internet connection. The common programming environment spares the students the tedious and, sometimes frustrating, task of configuring individual computers. Students require only Internet access to begin work on assigned assignments and projects. When a student submits a program through HTTP requests, the WCM server processes the requests and passes its processing requests to the application server, which is the Apache server in our system. The application server ships the execution of the program to the runtime environment, where the program is executed in a restricted environment (that is, environment with time and memory limits and other security restrictions). After execution, the output is sent back to the application server, from where it is passed on to the student's browser through the WCM server. The output typically contains the direct output from running the program, expected results, comments or suggestions if there are any deviations between the program output and the expected results, compiling errors if any, and grades. Because students receive immediate feedback when they are working the programming assignments, they are able to determine whether they are on the right track when devising solutions to a given homework problem. Most importantly, students can actually study the feedbacks they received and use them to improve their work.

THE ASSIGNMENT MANAGEMENT AND AUTOMATED GRADING AND FEEDBACK SYSTEM

The web-based assignment management and automatic grading and feedback generation system consists of five basic modules: editing tool, malicious code checking subsystem, database and file management system, runtime environment (with facility for compiling and running program against data and test cases and for applying restrictions), and auto-grading and feedback component. The rest of this section describes the components of the system, which are shown in Figure 1.

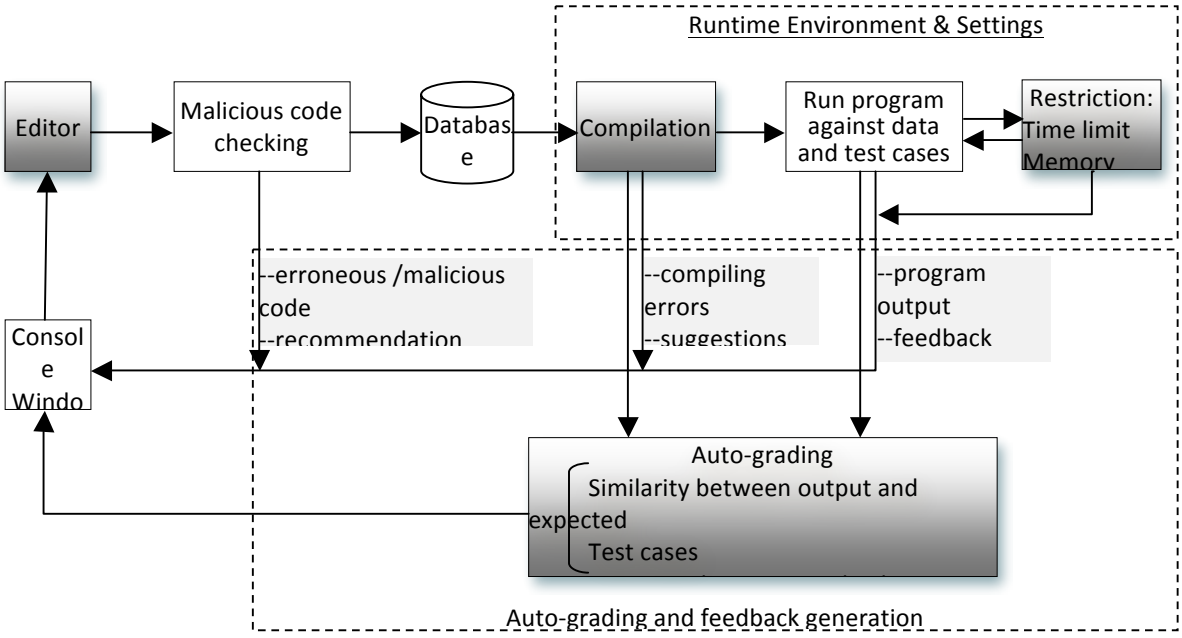

Figure 1. Components of the Assignment Management and Automated feedback System

Editor Tool

Integrated Java Development environments often require a high amount of computing resources, which may not be feasible on students' personal computers. Furthermore, there are hardly any free systems available that can be used to manage java programming courses in a distributed fashion. Thus, we embarked on developing the Java-Editor to allow students to work at home or from anywhere with Internet access. The editor provides an interface where users edit, compile, and run their java programs. Figure 2 shows the main windows of the Java-Editor for a student account; the toolbars and other input fields in the middle of the window allow the student to write, edit, compile and debug java programs. The editor tool simulates the command-based environment, where one may specify the arguments for the main method, read data files, or get input from keyboard.

Figure 2. Editor Tool-Student

Malicious Code Checking

It is anticipated that there will be cases where programs submitted by students will unintentionally or intentionally attempt to perform undesirable operations on the file system on the server. For instance, a malicious program may execute codes to access, view, modify or even delete some files in the system; a program may also go into an infinite loop. Even though these situations can be prevented by running the program in a "sandbox" mode, that will interfere with other features of the system. For example, if a program runs in a "sandbox" mode, sharing of class or data files with students, which is part of auto-grading and feedback generation process, is not possible. Thus, it should be possible to disallow specific operations on the file system in order to prevent malicious acts. Hence, our system does not allow a program to view or create a new file, and the system is configurable to limit the resources (including run time, number of threads, and memory allocation) that can be used by a program. If a code with potential harmful results is detected by the code checking component, the student is notified to make corrections before moving forward.

Database

The data base is the repository for managing all files in the system. There are three sub-modules: personal files, assignment files, and shared files in the public folder. The assignment management module handles all files that are associated with assignments. The instructor can use this module to share auxiliary files that are required to complete an assignment. Data files and testing-cases may be provided to allow students to test whether their work meet the requirements. Students can work in the assignment directory until the due date. No assignment submission is required because the grader can access students' work through the system when they are ready to be graded. The instructor designates a directory for a particular course section and all students enrolled in that section have access to it. The instructor can also post examples, class work or other files in the shared public folder for students review and testing. Figure 3 shows the three sub-modules for the file management system.

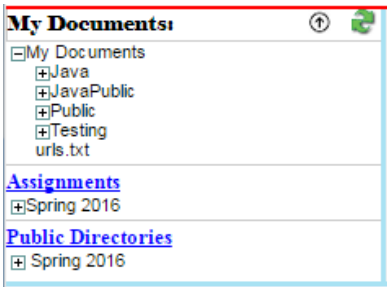

Figure 3. File Management

Runtime Environment and Settings

This module is responsible for compiling and running program against data and test cases and for applying restrictions. Java runtime environment is evoked to compile programs that have been found to contain no potential malicious codes. If syntax errors are found, the compiler error messages are sent to the auto-grading and feedback generator module, which in turn makes suggestions and sends them back to console window to aid debugging and corrections. After compilation, the program executable files are passed on to the Java runtime environment, where the required data streams and files are supplied. The system currently supports only Java program development and provides three standard ways of feeding programs with data: Passing arguments list to the main method, providing input from keyboard, and reading data files. After program execution, the output is sent to the auto-grading and feedback generator module, which in turn processes the output, including results from running test-cases, runtime exceptions, and output of the main program, and generates feedback to be sent back to console window.

The pre-compilation malicious code checking process is based on syntax and restriction rules. Therefore, the process cannot detect erroneous or malicious actions that may result from the execution of the program. For example, pre-compilation check for malicious code cannot detect infinite loops or memory leak, which may result in poor response from the server or in a crash of the server completely. To prevent these runtime erroneous or malicious behaviors, the system sets a limit on time and memory use for each program. When a program reaches the time limit or allocated memory, the restriction utility service terminates the program and sends a corresponding message to the user interface window.

Auto-Grading and Feedback Generation

The purpose of providing feedback is to enhance the learning process (Duncan, 2007; Taras, 2003). Feedback is one of the most powerful influences on learning and achievement (Getzlaf et al., 2009, Kitaya and Timperly, 2007). Thematic analysis (Getzlaf et al. 2009) revealed five major themes: student involvement/individualization, gentle guidance, being positively constructive, timeliness, and future orientation. Effective feedback is proposed to improve the learning process by reducing discrepancies between current understandings/performance and a desired goal (Getzlaf et al., 2009, Kitaya and Timperly, 2007, Duncan, 2007; Taras, 2003). To achieve this goal, the feedback provided must actually be reviewed, studied, and used by the student to revise the submitted assignment. However, it is commonly reported that students do not read the instructor's feedback comments (Duncan, 2007), especially when the feedback does not come in a timely manner. To ensure that feedback is being used to improve students' programming skills, it must be delivered in a timely and actionable manner. But the reality is that programming assignments are still managed manually in most classrooms; hence it may take a few days, or even a few weeks for graders to grade and provide feedback to students. By the time a student receives feedback, he or she could have forgotten the details of the program in question. The student may simply ignore the feedback. Furthermore, for feedback to be effective, it must be concrete, specific, and useful. It should provide actionable information which students can understand and follow to revise their work and reduce the discrepancies between expected outcome and the output of their program. Figure 4 shows the conceptual model of the feedback system, while figure 5 depicts the implementation model.

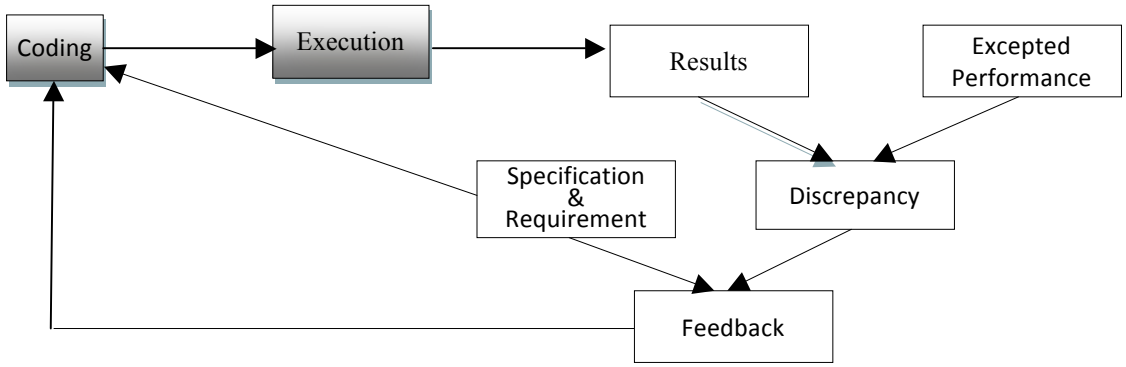

Figure 4. Conceptual Model of the Feedback System

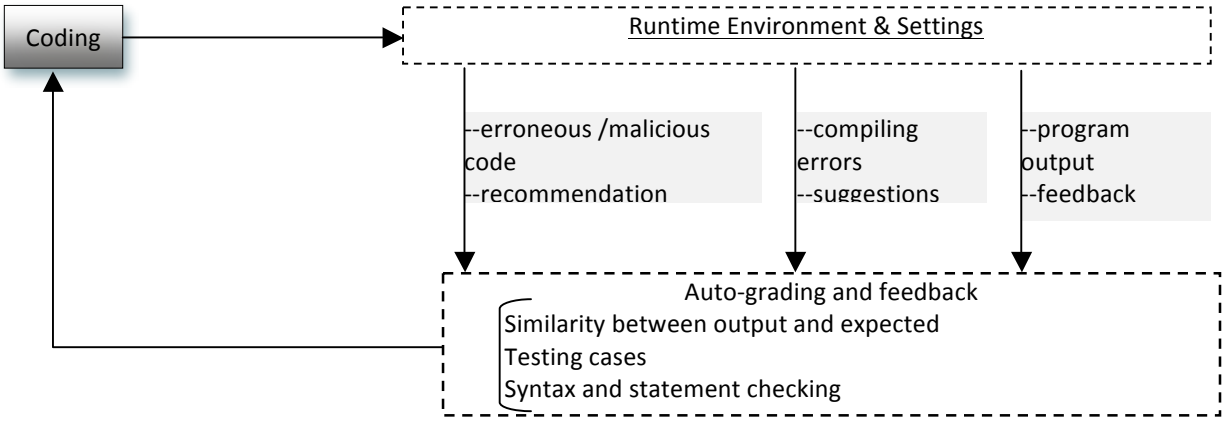

Figure 5. Implementation Model of the Feedback System

AUTOMATED GRADING AND FEEDBACK GENERATION PROCESS

The automated grading process consists of three stages that are organized in a hierarchical structure: Evaluating program output against expected results, assessing text cases, and evaluating syntax and statements. Figure 6 shows the stages in the hierarchical structure. The program, as a whole, is evaluated in the first stage; while in the second stage individual methods in the program are evaluated using the test cases approach. The third stage evaluates the program for syntax and statement correctness. At the first stage, the system runs the program against the supplied data (argument list for the main method, simulated keyboard input, or file input) and test cases. Then the output of the program is compared with the expected results based on their similarity. A 100% similarity means that the program output and the expected results are a perfect match. The score (the total possible score multiplied by the similarity percentage) is recorded in the gradebook.

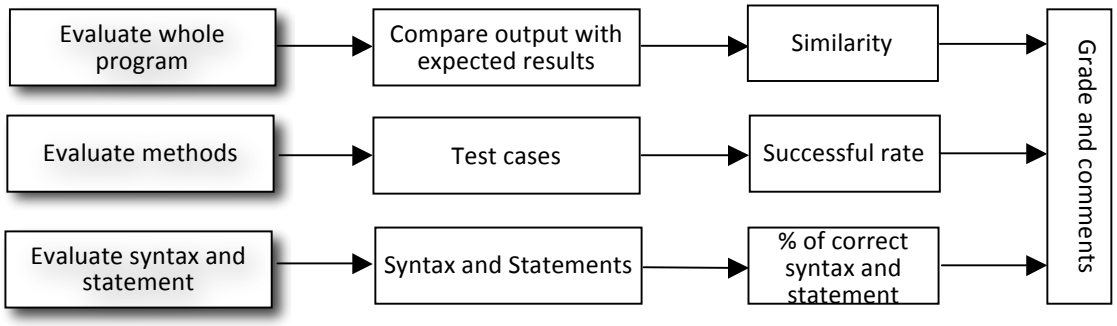

Figure 6. Auto-Grading Hierarchical Structure

Method evaluation is based on the well accepted test cases approach. In software engineering, a test case is a set of conditions under which one determines whether an application or software system (or one of its features) is working in accordance with its original design. The grading system uses the JUnit packages for assessing each method in a program. JUnit, a unit testing framework for the Java programming language, is one of a family of unit testing frameworks in test-driven development.

The system provides template that uses the paradigm of the testing framework: "first testing then coding". In practice, a set of testing cases or data file are provided to students for each assignment. Students start with the simplest case when they start coding, and gradually moving towards more complicated cases until meeting the full

requirements for a particular assignment. Let us consider an assignment which requires that students write a code to count how many times a word appears in a data file. The first testing data file may contain only one word. The code simply reads in the single word file and uses "if" statement to test whether the word is the same is the one in question. If the output differs from the expected result, the feedback "You may want to consider using *if* statement to test whether the word is the same as the one in the data file" is generated and sent back to the student. The second testing data file contains multiple words and students are expected to use a loop structure to test each word in the data file; and the third data file contains some words with mixed cases, or a work combined with punctuations. This process helps the student to make progress until all testing cases are tested.

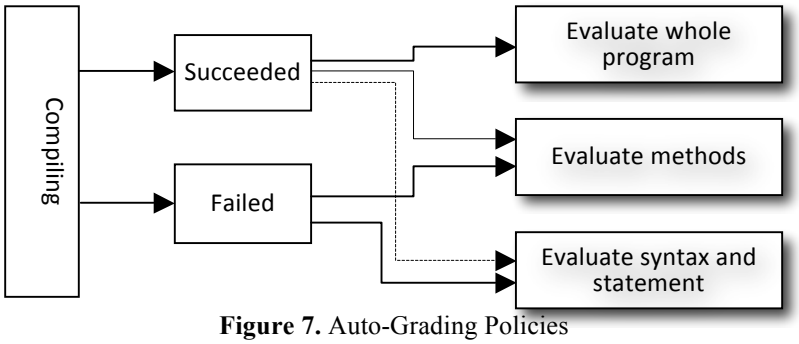

Figure 7. Auto-Grading Policies

When grading assignments, if a java program compiles successfully, most instructors use output similarity to evaluate the whole program. The instructor may choose not use test cases and/or syntax checking for more detailed evaluation. However, in our case, if a program failed to compile or runtime errors occur, we have chosen to use method evaluation and syntax checking so as to be able to award partial credits. This process requires grader involvement. However, since most students' work can compile and run properly with testing cases or data file, only a few submissions need be graded manually. Figure 7 illustrates the auto-grading policy for our courses. Solid lines represent the recommended auto-grading methods, and the dashed lines represent optional methods.

AUTO-GRADING AND FEEDBACK IN PRACTICE

In this section, we illustrate with examples how the feedback and hierarchical auto-grading structure is actually implemented in the system. The example was from an introductory java programming course after introducing control and loop structures. In this assignment, students were asked to write a program that reads a set of phone numbers from a file, parses each number, and transforms the phone numbers into the format: (area code) three digits-last four digits. (111) 111-1111 is a valid example. Three testing data files (data1.txt, data2.txt, and data3.txt) shown below were provided.

data1.txt

```
(804) 524-0001
(804) 524-0002
(804) 524-0003
```

data2.txt

```
8045240001
8045240002
8045240003
```

data3.txt

```
804-524-0001
8045240002
(804) 524-0003
```

Students started coding with the first data file, which is the simplest scenario. The first testing data file tests whether the student is able to write a code to read each line from a file and display the data without any further processing. Most students were able to perform this task. If a student's program failed to generate the expected output, the system generates the following message:

> You may consider using a *while* loop structure and read and display each line from the file until no more line to read from the file. For example, while(input.hasNext()) {…}

The second testing data file is designed to test whether the student can use "substring" method to reformat the phone number. A typical feedback message is as follows:

```
Use the substring method: use substring(startingIndex, endingIndex) to get the area code, the first three digits, and the last four digits.
```

```
Put area code inside parentheses; use a dash between the first three digits and the last four digits.
```

The assignment was graded using the top two tiers of the grading hierarchical structure: Evaluating the whole program based on the similarity between program output and expected output and assessing the use of the required method which takes a String as parameter and converts it to the required phone number format. Table 1 shows the test cases and expected results.

Table 1. Testing Cases and Expected Results

Method	Testing Cases	Expected Results
converts	converts("(804) 524-0001")	(804) 524-0001
	converts("(804) 5240001")	(804) 524-0001
	converts("804-524-0001")	(804) 524-0001.
	converts("8045240001")	(804) 524-0001
	converts("804 524 0001")	(804) 524-0001

The following data file was also used to evaluate the program as a whole:

```
804-524-0002
(804) 524 0002
8045240002
804 524 0002
```

Then a grade was calculated based on the percentage of the testing cases and similarity between outputs and expected results of the program. With auto-grading, the assignment for 28 students was graded in seconds, and each student received a customized comment on their gradebook. Figure 8 shows the comments generated automatically for a student.

```
Did not pass the following cases:
 converts("8045240002")
     Expected: (804) 524-0002
     Actual: 804 524-0002
     Fedback: Put area code inside parentheses
```

Figure 8. Sample of auto-grading comments

EMPIRICAL DATA

The web-based assignment management and auto-grading and feedback system has saved both students and instructors tremendous amount of time in past two semesters. In most academic units that offer programming courses, programming assignments are distributed either in paper form or through an LMS. Students download assignment shells or templates along with the instructional documents and proceed to work on the assignment on personal computers or in computer labs. Also, students must have a means of transferring program and data files if they need to work on different computers. But, as noted in a previous section, the manual process of downloading submitted program files for compilation and testing is cumbersome and time-consuming. It also has the effect of reducing the number of homework exercises that can be assigned. Furthermore, students are not able to determine whether they are on the right track when devising solutions to a given homework problem. When students need instant help in order to continue on a programming assignment, it is difficult, if not impossible, for them to get

immediate and effective assistance from instructors because the two parties (the student and the instructor) usually do not have simultaneous access to the relevant files. These problems do not contribute positively to the learning process; they only reduce the productivity of a course and the learning process (Demir, Soysal, Arslan, Yürekli, & Yılmazel, 2010). In reality, by the time a student receives feedback after a few days, or even a few weeks, the student could have forgotten the details of his/her program. The delay in giving feedback and grading can reduce the rate at which students acquire programming skills (Kitaya and Inoue, 2016). With the new system, students can check their work with the provided data files and testing cases. Most importantly, using the system, students can get instantaneous help from the instructor or peers if they encounter problems; this feature significantly improves students' performance and programming skills.

Table 2 and Table 3 show the data relating to two sections of the same course taught by the same professor. Serverside scripts were running on the server to record compiling errors. Table 2 depicts the errors (compiling errors and runtime errors) observed in the work submitted by students, as well as the overall performance, which was measured as the average grade for each section. Table 3 compared the time spent on assignment preparation and grading for the two sections. As shown in table 2 and table 3, with the new system, we are able to give more assignments and, at the same time, spent less time on preparing and grading the assignments.

Table 2. Evaluation of the Overall Performance of the System

Sections	Total number of assignments[1]	Total programming files	Total errors (compiling, runtime errors)	Errors rate per Assignment per student	Error rate per file	Average grade
Without using the system	15	33 x 20[2]	98	29.70%	14.85%	81
Using the system	25	50 x 22[2]	56	10.18%	5.10%	92

Table 3. Comparison of the Time Spent by the Instructor on Programming Assignments

Sections	Preparation per assignment[3]	Grading per assignment	Total per assignment	Time saving per Assignment per student	Total time spent[6]	Total saving
Without using the system	0	15[4][7]	15[7]	0	75 hrs	0
Using the system	3[7]	3[5][7]	6[7]	9[7]	60 hrs	15 hrs

[1] Including lab work, homework, project assignments.
[2] The number of students in each section.
[3] Relative time spent on preparation. Since both options need initial preparation, the new system has an extra time for setting up online assignments.
[4] The time spent on downloading, unzipping files, copying files to appropriate directory, compiling and running the program, providing feedback, recording grades, zipping files, and uploading zipped files to the LMS. This process applies to each submission. Additional time is not required if a submission runs perfectly.
[5]The time spent on only those submissions with errors. In addition to automatically generated comments, instructors may want to check and add more customized comments if he or she so desire. There is no need to look at those submissions with perfect scores.
[6] The time does not include the initial preparation for each assignment, as noted in [2], this portion is the same for both sections.
[7] Measured in minutes.

CONCLUSIONS AND FUTURE WORK

One of the best methods of learning in programming courses depends on practical exercises. But the process of preparing, collecting and grading homework manually takes a lot of time. When the number of students increases, the number of homework given tends to go down. As a result, the effectiveness of the course is diminished. One way to solve this problem is to distribute homework through an automatic grading system and provide fast feedback. Thus, automatic grading of programming assignments is an important topic in academic research. It aims at improving the level of feedback given to students and optimizing the professor's time. Empirical data confirmed that the accuracy of scoring is very high and the response time of the system is satisfactory. The experience in the classroom has been very positive and the system did increase the productivity of the class. A significant increase in the quality of students' codes and a decrease in compilation and runtime errors have been observed. Moreover, we are working on adding new features to the system to provide features such as plagiarism check and integration of online office hours with the aim of providing live discussion on assigned computer programming work. This study is an initial effort to evaluate the impact of the online programming assignment management system and auto-grading on the learning experience in introductory courses. Although it provides a number of interesting results, the scope of the study was limited and further research may be undertaken to assess the effectiveness of the system from different aspects, including the quality of student interaction with the system, impact on the programming performance of the instant feedback on the assignment, as well as how the feedback generated by the auto-grading can be used to improve the learning process.

REFERENCES

Alamo, J.M., Alonso, A., & Cortés, M. (2012). Automated Grading And Feedback Providing Assignments Management Module, *ICERI2012 Proceedings, 5th International Conference of Education, Research and Innovation*. November 19-20, pp. 3784-3793, Madrid, Spain.

Beierle, C., Kula, M., & Widera, M. (2003). Automatic analysis of programming assignments. *DeLFI Proceedings, e-Learning Fachtagung Informatik* (DeLFI), pp. 144-153.

Brenda, C., Kurnia, A., Lim, A., Oon, W. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education 4,* pp.121–131.

Caiza, J., Del Alamo, J. (2012). Programming Assignments Automatic Grading: Review of Tools and Implementations. *Inted 2013 Proceedings*, pp. 5691–5700.

Demir, Ö., Soysal, A., Arslan, A., Yürekli, B., & Yılmazel, Ö. (2010). Automatic Grading System for Programming Homework. *Computer Science Education: Innovation and Technology, CSEIT*.

Duncan, N. (2007). „Feed-forward": improving students" use of tutor comments. *Assessment & Evaluation in Higher Education*. *32*(3), pp. 271 -283.

Dutton. J.C. (2001). WebAssign: A better homework delivery tool. [Online]. Available: http://technologysource.org/article/webassign/

Getzlaf, B., Perry, B., Toffner, G., Lamarche, K., & Edwards, M. (2009). Effective Instructor Feedback: *Perceptions of Online Graduate Students. Journal of Educators Online*, *6*(2).

Hattie, J. and Timperley, H. (2007). The Power of Feedback. *Review of Education Research. 77*(1), pp. 81-112.

Kitaya, H. & Ushio Inoue, U. (2016). An Online Automated Scoring System for Java Programming Assignments, *International Journal of Information and Education Technology*, *6*(4).

Roberts, G.H.B. and Verbyla, J.L.M. (2003). An Online Programming Assessment Tool. In Proc. *Fifth Australasian Computing Education Conference (ACE2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, 20. Greening, T. and Lister, R., Eds., ACS.* pp. 69-75.

Singh, R., Gulwani, S. & Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. *PLDI Procceedings. 34th ACM SIGPLAN conf. on Programming Language Design and Implementation (PLDI)*, pp. 15-26.

Taras, M. (2003). To feedback or not to feedback in student self-assessment. *Assessment and Evaluation in Higher Education, 28*(5), pp. 549- 565.

Thiébaut, D. (2015). Automatic Evaluation Of Computer Programs Using Moodle's Virtual Programming Lab (VPL) Plug-In. *Consortium for Computing Sciences in Colleges (CCSCNE) 2015*, *Boston.*