An Exercise Management System for Teaching Programming

Sho-Huan Tung, Tsung-Te Lin, and Yen-Hung Lin
National Yunlin University of Science and Technology / Department of Information Management, Yunlin, Taiwan
Email: {tungsh, lintt, lyhcode}@yuntech.edu.tw

***Abstract*—An effective learning activity in a computer programming course is to study and practice computer programs. In order to help students to submit exercises and to assist instructors to mark programming exercises, a number of program submissions and assessment systems have been developed. However, these systems do not provide sufficient support for instructors to design exercises that can help students to study and practice computer programs in an incremental manner. With the primary aim to improve the teaching and learning of computer programming, we have developed a programming exercise management system, namely Programming Learning Web (PLWeb), to assist instructors to design computer programming exercises and to help students to study and practice programming exercises. PLWeb provides an integrated development environment (IDE) which is used not only as an authoring tool for instructors to compose exercises but also as a novice-friendly editor for students to study programs and to submit solutions. In addition, PLWeb allows instructors to use visualized learning status to assist students with difficulties. A plagiarism detection tool is also provided to deter students from plagiarism.**

***Index Terms*—programming learning tools, programming exercises design, computer science education, architectures for educational technology system, plagiarism detection**

I. Introduction

Programming is not an easy subject for beginners to study [1]. Besides syntax and semantics, beginning programmers also face the challenge of learning abstract concepts, as well as testing and debugging techniques to be able to solve problems. Without proper assistance, it can be very difficult for students to overcome all of these challenges.

In a survey on the difficulties of novice programmers [2], Lahtinen et al. suggested that "learning by doing" should be a part of studying programming at all times. Their survey reported that both students and teachers agreed that practical learning situations were the most useful. In particular, exercise sessions were rated more useful than lectures, and practical sessions in computer rooms as well as programming by themselves were rated more useful than studying by themselves. In addition, example programs were considered as the most useful type of material. However, novice learners still need assistances in order to overcome erring, floundering, and a variety of problems encountered while practicing programming [3].

To compensate this, van Merrienboer proposed a completion strategy as the basis of a programming tutor for beginners to assist their learning [4, 5]. The completion strategy uses commonly used programming idioms, patterns, or well-designed programs and their partially completed versions as model programs for students to finish, modify and extend. For example, students may be required to finish a partially completed count-controlled-loop by fill-in-the-blanks for its count control variables before attempting to solve other similar problems. This strategy forces students to study and analyze the model programs so that they can imitate them properly. The benefits of the completion strategy have been demonstrated in [4, 5].

In addition to exercises that support the completion theory, other types of programming exercises such as multiple-choice, debugging, and tracing program execution can also help students to comprehend the theory, syntax, semantics, and behaviors of programs.

In order to support learning programming by doing exercises, the instructor has to prepare many kinds of programming exercises for students to practice. A programming exercise has several components: a problem statement, a solution, test cases, and optionally a partial or buggy version for students to complete or debug. The instructor also needs to package and upload them to a server to deliver to students. These tasks added too much workload to the instructor [6]. An adequate tool is needed to help instructors to compose and students to practice exercises to learn programming.

The typical steps associated with the preparation and processing of exercises are the following: composing and distributing exercises, writing and submitting solutions, and marking submissions and returning feedback [7]. These steps can be divided into two phases: the preparation phase and the processing phase. The preparation phase includes the steps of composing and distribution of exercises. The other steps can be subsumed in the processing phase. A wide variety of tools have been developed to support the processing phase of programming exercises, but few tools are available to support the preparation phase [7].

In this context, PLWeb was designed to satisfy the needs of both the preparation and processing phase of programming exercises. A unique design of PLWeb is a dual-purpose IDE which is used not only as an authoring tool for authors to produce exercises but also as a novice-friendly editor for students to write, test, and submit

doi:10.4304/jsw.8.7.1718-1725

solutions. Furthermore, the system allows instructors to visualize students' learning status in order to actively provide assistance to needed students. A plagiarism detection tool is also developed to deter students from plagiarism.

II. Related Works

Over the years, a number of program submissions and assessment systems that put emphasis on the processing phase of programming exercises have been developed [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]. The majority of the systems verify the correctness of a student's program based on its outputs. They compare the outputs generated by a student's program and the expected outputs provided by the teacher.

Although the method is simple and popular, some researchers have offered additional features to improve the testing process. RoboProf [8, 9] uses a set of test data that may be generated randomly in order to prevent students from tricking the system by writing code to print the appropriate output. CourseMarker [10] compares the textual outputs by matching to regular expressions that define the expected outputs. VIOPE [11, 12, 13] can check whether the student's program is done using a certain structure, e.g. for-loop. PASS [6, 14] allows teachers to store some predefined comments with some specific patterns of mistakes, and the system can show the comments to the students to help the students to debug the program. BOSS [15] does not rely solely on testing through textual outputs but also on the JUnit testing mechanism when Java is the language used, and a similar design can also be found in Oto [7] and Marmoset [16]. Oto [7] provides instant feedback to students while testing their programs before the final submission. Marmoset [16] does not allow students to access the instructor's private test cases in order to encourage students to start their work early and to think critically about their work. Web-CAT [17] emphasizes correctness and completeness of testing which can be judged in comparison to a reference set of tests provided by instructors.

Some of the systems allow instructors to customize the marking process or to extend the assessment features in order to provide more flexibility and detailed marking results. CourseMarker [10] allows instructors to customize the marking process using a file that contains Java code and can access CourseMarker's state or call external tools. CourseMarker incorporates a variety of marking components such as typography, testing, specific feature inspection, and plagiarism detection. Web-CAT [17] allows the marking process to be customized using XML configuration files, and it incorporates a number of components to check commenting conventions, adherence to coding style guidelines, and use of potentially bug-inducing coding practices. The marking process of Oto [7] can be customized and the system can be extended with various marking components such as testing, style, structure, etc.

In addition to programming exercises, some of the systems also provide multiple-choice questions for students [10, 12, 15].

A combination of various systems can also be found in the literature. For example, EduJudge [18] integrates a submission system with automated evaluation into the Moodle e-learning platform and a competitive learning tool. It shows that the system can motivate students and improve students' academic outcomes.

These above systems are server-based, since they require students to submit their programs to a server for assessment and receive feedback from it. The assessment mechanism of PLWeb differs from the server-based assessment systems in that PLWeb's IDE integrated assessment with compiling and execution steps and runs on the user's computer. In addition, the IDE can load several exercises at a time for students to practice them one after another without delay. This feature makes PLWeb easier and more efficient to use than server-based assessment systems.

Besides assessment, some program submission systems also contain plagiarism detection tools for detecting plagiarism. CourseMarker [10] and BOSS [15] call for the help of external plagiarism software to detect program submissions that are similar. RoboProf [8, 9] detects plagiarism by adding an identifying watermark at the end of the main method in a Java source file. A technique for detecting plagiarism is also found in PLWeb. Unlike aforementioned techniques, PLWeb determines plagiarism suspects by analyzing the testing history and the editing events collected while students are editing their programs.

Although most of the program submission systems focus on automating the processing phase, a few systems also support the preparation phase. In PASS [6, 14], the test cases are grouped into three levels of difficulty for various ability levels. EduJudge [18] provides a problem database with different levels of difficulty in order to help students learn progressively. Both PASS and EduJudge provide exercises at different levels of difficulty to allow students to solve problems pertaining to their corresponding ability levels, but they do not provide tools to support exercise authors to create programming exercises conveniently. PLWeb, however, provides exercise authors an authoring tool to create multiple-choice, debugging, output predication exercises as well as exercises that support the completion strategy.

In addition, PLWeb provides better support for teaching programming in computer classrooms. Most of the program submission systems allow instructors to monitor students' progress. PLWeb improves the monitoring feature by illustrating not only students' progress but also the time spent on each exercise. Via a visualization tool, the instructor can locate students' difficulties that they encountered while in a laboratory session. PLWeb also provides the instructor with snapshots of students' codes which are captured every time when students test their programs (this similar feature can also be found in Marmoset [16]). These snapshots help the instructor to understand students'

difficulties. If the instructor encounters many students having similar problems, he or she can decide whether or not to provide additional short lectures on the subject to all students.

III. Using PLWeb in a Computer Classroom

Although PLWeb can be used as a self-learning tool, some of its features are designed for instructors to teach programming in a computer classroom. After using PLWeb, we experienced a change of our teaching from presenting lectures to preparing and assisting students in writing exercises. Usually a short lecture is given at the beginning of a class. Following that, students are allowed to download exercises to practice. Several types of exercises can be downloaded and practiced using the IDE. The first few exercises are usually multiple-choice questions meant to test students' understanding of the lecture material just presented. These may be followed by exercises that require students to correct a buggy program, to predict the output of a program, to complete a partial program, to extend an existing program, and to write new programs.

Students are motivated by giving them immediate feedback using the compiling, testing, and assessment tools integrated with the IDE and a pie chart displaying their learning progress on the web browser (see Figure 1). Exercises not finished during class time are given as homework assignments and need to be submitted before a certain due date set by the instructor. After the due date, solutions of the assigned exercises are posted on the web.

While students are practicing, instructors can monitor students' learning status and provide assistance to those students needing help. Students are encouraged to help one another while working on exercises. However, in order to discourage students from copying other students' solutions, PLWeb provides a plagiarism detector to deter students from plagiarism. The plagiarism detector displays a warning message when it finds any suspicious plagiarism.

IV. System Structure

PLWeb has two main components--the server and the IDE (see Figure 2). The IDE can be downloaded from the server when the user clicks a button to start practicing or composing associated with a learning unit. It can also

Figure 1. Students can see their learning progress by a pie chart.

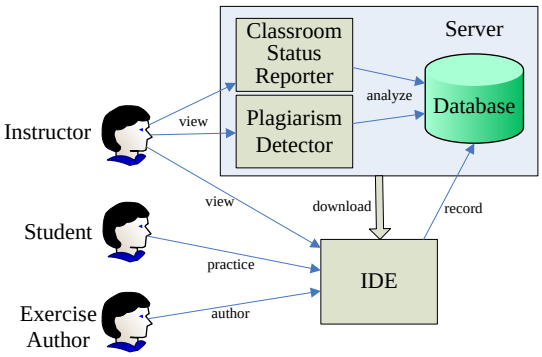

Figure 2. System structure overview.

record selected editing events and testing results and store them into the server's database. The server has two sub-systems: the classroom status reporter and the plagiarism detector. Both of them use information in the database to produce students' learning status and to detect plagiarism.

A. The IDE

The IDE is a modified version of jEdit [19] – an open source program editor written in Java. jEdit provides features indispensable for writing programs such as "automatic indentation" and "bracket matching." In PLWeb, several additional plug-in components are embedded in jEdit to help exercise authors/instructors and students to perform their tasks.

The IDE consists of four areas (see Figure 3): the description area (top-left), the editing area (top-right), the test sample area (bottom-left), and the testing area (bottom-right). The description area is a mini browser, which displays problem statements written in HTML. For an exercise with a partial solution, the file associated with the partial solution is shown in the editing area. The student can complete the exercise by extending or filling in code in the editing area.

The process of using the IDE is shown in Figure 4. When a student clicks the Exercises button on a web page, the IDE is started with a set of exercises downloaded from the server using the Java Web Start technology [20]. The student can read the exercise description and starts practicing using the program in the editing area. This program may be a partial program, a buggy program, or a program that interact with the student to answer multiple-choice questions. Upon clicking the execute button, the IDE calls a pre-installed compiler in the student's

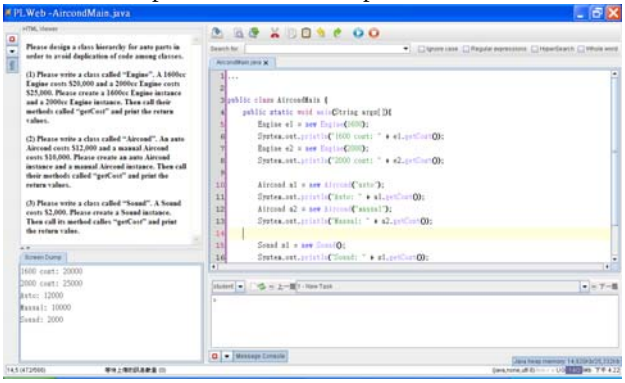

Figure 3. The IDE.

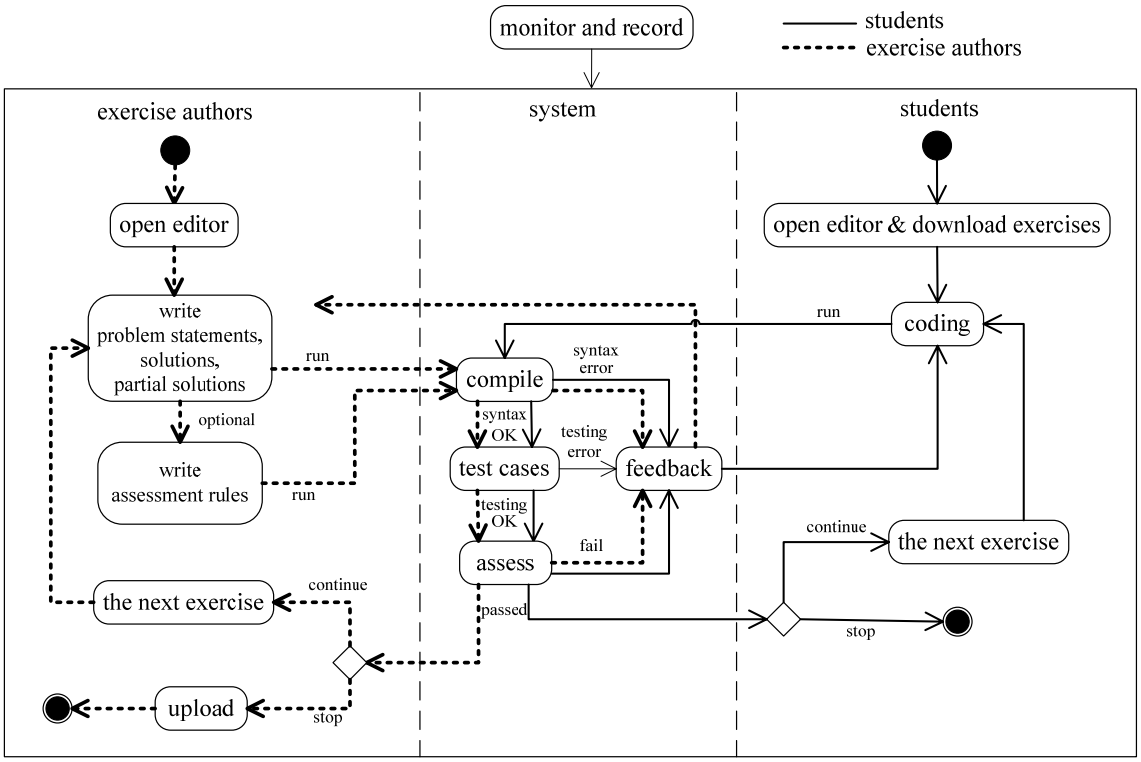

Figure 4. The process of using the IDE.

computer to compile and execute the program in the editing area. The student can test the program by typing its input data using information provided in the test sample area. Whenever the student finishes testing a program, the program and the test results are sent to the server automatically.

For most exercises, a comparison between the outputs of the program and the contents in the test sample area is sufficient to determine whether the program has passed the test. For more involved problems, the exercise author may choose to provide additional assessment rules to further assist the assessment.

Assessment rules are written in XML. Figure 5 shows a fragment of an example assessment rule. The <patterns> element defines a set of specific patterns, which contains one or more <pattern> element(s). Each <pattern> element designates a particular pattern represented by regular expressions. The assessment rules in Figure 5 mean that a student's program must contain exactly one occurrences of the keyword "for" in the Account class in order to pass the check. This feature is useful when students are required to write a "for" loop rather than other types of loop to solve a problem. The assessment tool can also perform checks on indentation and existence of comments in students' programs in order to improve the quality of students' code.

```
...
<patterns>
    <pattern>
        <block-begin><![CDATA[class(\s*)Account(.*)\{]]></block-begin>
        <block-end><![CDATA[}]]></block-end>
        <keyword><![CDATA[for]]></keyword>
        <operator><![CDATA[=]]></operator>
        <frequency><![CDATA[1]]></frequency>
    </pattern>
    ...
</patterns>
```

Figure 5. A fragment of an example of assessment rules.

For Java exercises, testing programs written in JUnit can also be used to assess students' solutions.

The textual comparison mechanism also supports assessment of multiple-choice and output prediction questions. A complete program that prints the description of the question and waits for students to enter the answer can be loaded in the editing area. Executing the program displays the question in the command shell and students can answer the questions by typing their answers. For these kinds of questions the test sample area is left empty.

After finishing writing and testing an exercise, the student can click the right arrow (next) button in the IDE to start working on the next one. This feature allows the student to work on one exercise followed by the next without delay.

Different from server-based program submission systems which require several steps from the user to compile/execute a program, submit it, and wait for the assessment results, PLWeb's IDE compiles, executes, and assesses tasks on the client's computer by clicking a single button. This design not only reduces the load and complexity of the server but also makes PLWeb easier and more efficient to use than the server-based systems. As a result, students can practice a set of exercises step by step to learn programming in an incremental manner. In addition, the partial solutions for some of the exercises further help students to focus on primary concepts without being distracted by details of less important program statements.

Creating exercises requires even more supporting features than practicing exercises. When the IDE is placed in the authoring mode, it opens three file-tabs in the editing area. The exercise author can edit any one of the following three areas: 1) problem statements, 2) solutions, or 3) partial solutions by switching these three tabs. The exercise author can also optionally set assessment rules using a fill-in form. By clicking the execute button, the exercise author can test the solution program by entering its input data into the command shell. The content of the command shell is automatically saved for students to use as test sample. Upon finishing creating one exercise, the exercise author can click on the right arrow button to edit the next one. After finishing producing all the exercises, the exercise author can click the upload button to package and upload the set of exercises to the server.

Without the IDE, exercise authors need to use several different tools to create, package, and upload description files, solution files, partial solution files, screen dump files, and assessment rule files of exercises to the server. The IDE helps exercise authors to perform these tasks and reduces their workload tremendously, thus enable them to produce exercises that fulfill "learning by doing."

B. The Classroom Status Reporter

While students are practicing, instructors can see a bird's-eye view of learning status of an entire class via a visualization tool called the classroom status reporter (see Figure 6). The classroom status reporter displays the time spent on each exercise for each student using a colored bar chart which uses yellow, green, dark green, and red to represent the learning state as editing, completed, late, or error respectively. The height of each colored bar represents the time spent on each exercise.

By observing the colored bar chart, the instructor can obtain information such as students' programming speeds, as well as correctness, or problems encountered while working on an exercise. For example, a student having many short green bars can be identified as a fast problem solver with good precision, and a student having many long red bars is a student encountering learning difficulties. In addition, the instructor can open a snapshot of a student's code by clicking on a color bar in order to understand the student's difficulties. This information helps the instructor to decide whether to modify an exercise or to provide additional hints to all students or only to those students needing assistance.

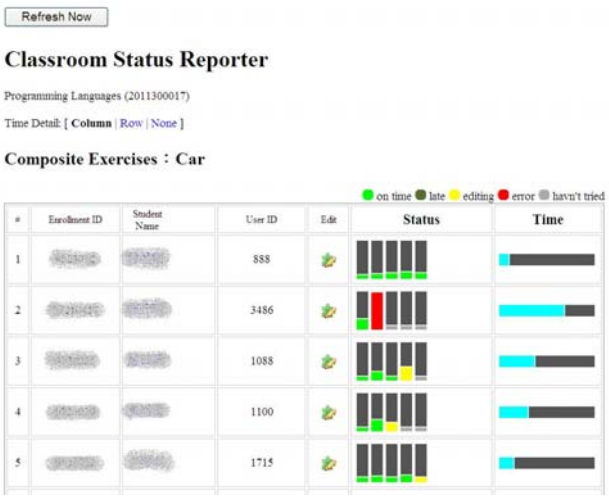

Figure 6. The classroom status reporter.

C. Plagiarism Detector

Learning by doing is the core concept of PLWeb. It is therefore important for students to practice programming by themselves.

Students who work honestly and students who plagiarize should have different forms of behavior. The plagiarism detector uses this fact to determine plagiarism suspects by analyzing the testing history and the editing events. For example, if the number of keys typed by a student to finish an exercise is lower than a reasonable range, then he/she may have plagiarized by copying and pasting another student's program. Similarly, if the amount of time spent by a student to finish an exercise is too short, then he or she might also have plagiarized. In normal situations, it is difficult to finish a program without going through several compile-execute-debug cycles and such cycles can be determined by the number of tests attempted and the modification keys (such as backward, forward, delete, etc.) typed. If a student does not go through such cycles, then he or she may also have plagiarized.

The plagiarism detector uses statistics to determine a reasonable range for normal situations. Values out of the reasonable range are outliers. Two methods are employed to detect outliers: Z-Score and Box-Plot. A normal Z-Score is derived by subtracting the population mean from an individual raw value and dividing the difference by the population standard deviation. However, computing the normal range usually returns a Platykurtic distribution in the situation, which is inappropriate to detect outliers. In order to obtain an approximate standard normal distribution, a logarithm is adopted. The modified formulas are

$$\overline{X} = \frac{\sum_{i=1}^{n}(\log_2 X_i)}{n} \tag{1}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(\overline{X} - \log_2 X_i)^2}{n}} \tag{2}$$

$$fnc = 2^{(\overline{X} - p \bullet \sigma)} \tag{3}$$

Where $\overline{X}$ denotes the mean; n denotes the number of students; σ denotes the standard deviation; and fnc is the lower fence; p is an adjustable parameter. A smaller p can reveal more outliers. In the plagiarism detector, p can be 1.0, 1.5, 2.0, 2.5, or 3.0, the default value is 1.5. If Xi is less than fnc, then Xi is an outlier.

On the other hand, Box-Plot detects outliers by medians and quartiles. The formula is

$$fnc = LQ - p \bullet (UQ - LQ) \quad (4)$$

Where LQ denotes the lower quartile; UQ denotes the upper quartile; p is an adjustable parameter. A smaller p can reveal more outliers. In the plagiarism detector, p can be set to 1.0, 1.5, 2.0, 2.5, or 3.0 with 1.5 as the default value. If Xi is less than fnc, then Xi is an outlier.

Instructors can choose the modified Z-Score or Box-Plot to detect outliers according to characteristics of exercises. If some exceptionally large values exist, instructors should choose Box-Plot to palliate the impact of extreme values. Otherwise, the modified Z-Score works well. Additionally, instructors can adjust the p value depending on the value of standard deviation and the strict level to detect plagiarism. If the standard deviation is large or the strict level is lower, then the instructor can choose a larger p value.

The plagiarism detector displays a "Warning" when it finds an outlier in one or more of the Time-spend, Error, Key-Stroke, or Modification Keys categories. If a student wants to dodge the detection, he/she must imitate the acts of normal practice. Imitating those acts is not only difficult but also time consuming. As a result, students are discouraged from plagiarism with the help of the system.

V. Evaluation

Two evaluation instruments were used in this study: 1) an experiment, which analyzes the effects of PLWeb on students' learning outcomes, and 2) a survey, which measures students' satisfaction on their experiences using PLWeb.

A. The Experiment

The experiment took place in 2009 for two sessions of a required C programming course offered in the freshman year by the Department of Electronic Engineering at National Yunlin University of Science and Technology in Taiwan. The two sessions were taught by the same instructor in a computer classroom with over 50 personal computers and their course contents were identical. The two sessions were allocated 3 hours per week and the numbers of students were 51 and 31 respectively. One session was assigned as the experimental group, and the other was assigned as the control group.

Students taking the course were selected by a highly competitive national entrance exam, and some of them may not know that they are not interested in or suitable for programming before being admitted. This is quite different from other experiments on program submission systems conducted in universities where every student of the university may choose to take an introductory programming course with hundreds of students before declaring their major.

The first exam of the course, the pre-test in the experiment, was held in the fifth week. After examining the scores by t-test, the results revealed that there was no significant difference between the two groups on participants' programming abilities (p=0.33). The participants were classified as having low, medium, or high programming ability based on the scores of the pre-test.

The experiment was held in the sixth week. The objective of the lesson was for students to understand the usage of pointers. Before the experiment, it was discerned that none of the participants had ever learned about pointers, but that they were already familiar with PLWeb since they had used PLWeb for five weeks. The experimental group used PLWeb and the control group did not use PLWeb. A 30 minutes exam, the post-test, was conducted before the end of the lesson. It is difficult to conduct experiments that take a longer period of time, since it is unfair for students who are unable to use PLWeb to assist learning.

Table 1 shows the mean (M) and the standard deviation (SD) of the scores obtained by the post-test. The students who used PLWeb achieved significantly better outcomes than those who did not use it (p=0.034).

TABLE 1.
COMPARATIVE OF THE POSTTEST RESULTS.

	Experimental Group			Control Group			T-Test p value
	n	M	SD	n	M	SD	
High	13	90.8	13.8	8	85.0	12.0	0.171
Medium	27	79.6	13.4	15	66.0	19.9	0.006**
Low	11	25.5	19.2	8	17.5	17.5	0.184
Total	51	70.8	28.5	31	58.4	30.9	0.034*

* Results are significantly different at p<0.05(T-Test).

** Results are significantly different at p<0.01(T-Test).

A deeper analysis shows an interesting finding. The students with medium programming ability who used PLWeb achieved significantly better outcomes than those students in the same level but who had not used it (p=0.006). However, the students with high or low programming abilities who used PLWeb had better outcomes than those students in the same levels who did not use PLWeb, but it has not yet achieved a significant level (p>0.05). The reason for this phenomenon is probably that students with a high programming ability are less affected by learning tools, and students with a low programming ability may not be interested in or suitable for programming.

B. The Survey

The evaluation surveyed students who used PLWeb for 1 or 2 semesters during 2010 to 2011. A total of 203 students answered the questionnaires. Responses were received from 3 universities in Taiwan: National Yunlin University of Science and Technology (78 respondents), National Formosa University (57 respondents), and Lin-Tung University of Technology (68 respondents). Most (85.3%) of the students taking part in the survey had

TABLE 2.
QUESTIONS AND RESULTS IN THE STUDENTS' QUESTIONNAIRES.

Code	Question	Result
S1	Which of the following types of materials is most helpful to you?	Lecture Notes: 8 (3.9%) Slides: 6 (3.0%) Textbooks: 10 (4.9%) Program examples, exercises, and solutions: 179 (88.2%)
S2	Listening lectures is helpful to you.	AVG: 3.52 STDEV: 1.02
S3	Practicing exercises without PLWeb is helpful to you.	AVG: 3.43 STDEV: 0.93
S4	Practicing exercises with PLWeb is helpful to you.	AVG: 4.10 STDEV: 0.85
S5	Given the same amount of time, using PLWeb allows you to practice more exercises than otherwise.	Disagree: 16 (8.3%) Agree: 187 (91.7%)
S6	Given the same amount of time, using PLWeb to learn programming is more effective than not using PLWeb.	Strongly Disagree: 2 (1.0%) Disagree: 3 (1.5%) Neutral: 42 (20.7%) Agree: 95 (46.8%) Strongly Agree: 61 (30.0%)

experience in programming before using PLWeb.

The questions and results of students' questionnaire are shown in Table 2. Most of the respondents (88.2%) considered that program examples, exercises, and solutions (S1) were the most useful type of material.

The purpose of statements S2~S4 is to compare the helpfulness among different learning styles. The statements asked the respondents to evaluate on a five-point scale from very unhelpful (1) to very helpful (5). The respondents rated practicing exercises using PLWeb (S4) more helpful than listening lectures (S2) or practicing exercises without PLWeb (S3). Furthermore, examining paired t-test results on S2 and S4 as well as S3 and S4 shows that both p values are much less than 0.01.

Statements S5 and S6 asked respondents to compare the learning effectiveness between using PLWeb and not using it. Most respondents (91.7%) agreed that in the same amount of time, using PLWeb allowed them to practice more exercises than otherwise, and most respondents (76.8%) agreed or strongly agreed that PLWeb is an effective tool for learning programming.

We also asked six instructors who used PLWeb for at least one semester about their experiences in using PLWeb. Four of the instructors indicated that they provided over 80% of the time to students for practicing exercises or discussing in class. And the other two instructors provided over 70% and 60% of the time for practicing exercises, respectively. All of the six instructors agreed that PLWeb is helpful with teaching programming courses and would continue to use PLWeb in the following semester.

VI. CONCLUSION

PLWeb allows the teaching and learning of computer programming using the "learning by doing" approach. By progressively practicing a series of pedagogic exercises step by step, the steep learning curve is effectively eased. The results of the surveys revealed that students like PLWeb since they regard it as helpful, effective, and efficient in learning programming.

The IDE not only improves students learning but also allows exercise authors to apply learning strategies on designing exercises efficiently. The classroom status reporter allows instructors to monitor students' learning status and to actively assist students with difficulties. In addition, the plagiarism detector can deter students from plagiarism.

PLWeb currently provides exercises and learning materials for Scheme, C, and Java. In addition to self-learning, it is a helpful tool for teaching programming in the computer classroom and has been successfully used in several universities in Taiwan.

ACKNOWLEDGMENT

This project is partly supported by an educational improvement fund from the Ministry of Education of the Republic of China. The authors would like to thank many students who have participated in the development of PLWeb.

REFERENCES

[1] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys*, vol. 37, pp. 83-137, 2005.

[2] E. Lahtinen, K. Ala-Mutka and H.-M. Järvinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bulletin*, vol. 37, pp. 14-18, 2005.

[3] D. C. Merrill, B. J. Reiser, S. K. Merrill and S. Landes, "Tutoring: Guided learning by doing," *Cognition and Instruction*, vol. 13, pp. 315-372, 1995.

[4] J. J. G. V. Merriënboer, "Strategies for programming instruction in high school: Program completion vs. Program generation," *Journal of Educational Computing Research*, vol. 6, pp. 265-285, 1990.

[5] J. J. G. V. Merrienboer and M. B. M. D. Croock, "Strategies for computer-based programming instruction:

Program completion vs. Program generation," *Journal of Educational Computing Research*, vol. 8, pp. 365-394, 1992.
[6] Y. T. Yu, C. K. Poon, and M. Choy, "Experiences with PASS: Developing and using a programming assignment assessment aystem," *In Proceedings of the Sixth International Conference on Quality Software*, Beijin, China, pp. 360-368, 2006.
[7] G. Tremblay, F. Guérin, A. Pons and A. Salah, "Oto, a generic and extensible tool for marking programming assignments," *Software: Practice and Experience*, vol. 38, pp. 307-333, 2008.
[8] C. Daly and J. M. Horgan, "An automated learning system for Java programming," *IEEE Transactions on Education*, vol. 47, pp. 10-17, 2004.
[9] C. Daly and J. Horgan, "A technique for detecting plagiarism in computer code," *The Computer Journal*, vol. 48, pp. 662-666, 2005.
[10] C. A. HIGGINS, G. GRAY, P. SYMEONIDIS and A. TSINTSIFAS, "Automated assessment and experiences of teaching programming," *ACM Journal of Educational Resources in Computing*, vol. 5, pp. 1-21, 2005.
[11] U. Nikula, O. Gotel and J. Kasurinen, "A motivation guided holistic rehabilitation of the first programming course," *ACM Transactions on Computing Education*, vol.11, issue 4, article no. 24, 2011.
[12] E. Vihtonen and E. Ageenko, "VIOPE - computer supported environment for learning programming languages," *In Proceedings of International Symposium on Technologies of Information and Communication in Education for Engineering and Industry*, pp. 371-372, Lyon, France, 2002.
[13] J. Carver and L. Henderson, "Viope as a tool for teaching introductory programming: An empirical investigation," *In Proceedings of the 19th Conference on Software Engineering Education and Training*, Turtle Bay, Hawaiian, pp. 9-16, 2006.
[14] F. L. Wang and T. L. Wong, "Designing programming exercises with computer assisted instruction," *Lecture Notes in Computer Science*, vol. 5169, pp. 283-293, 2008.
[15] M. Joy, N. Griffiths and R. Boyatt, "The boss online submission and assessment system," *ACM Journal of Educational Resources in Computing*, vol. 5, pp. 1-27, 2005.
[16] J. Spacco, et al., "Experiences with Marmoset: Designing and using an advanced submission and testing system for programming courses," *ACM SIGCSE Bulletin*, vol. 38, pp. 13-17, 2006.
[17] S. H. Edwards and M. A. Perez-Quinones, "Experiences using test-driven development with an automated grader," *Journal of Computing in Small Colleges*, vol. 22, pp. 44–50, 2007.
[18] E. Verdú, L. M. Regueras, M. J. Verdú, J. P. Leal, J. P. d. Castro and R. Queirós, "A distributed system for learning programming on-line," *Computers and Education*, vol. 58, pp. 1-10, 2012.
[19] S. Pestov, J. Gellene and A. Ezust, "jEdit 4.5 user's guide," http://www.jedit.org/users-guide/index.html, accessed: September 2012.
[20] Oracle, "Java Web Start guide," http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html, accessed: September 2012.

Sho-Huan Tung is a professor in Information Management Department at National Yunlin University of Science and Technology, Taiwan. He received his Ph.D. in Computer Science from Indiana University, USA in 1992. His research areas are programming learning environments and program visualization systems.

Tsung-Te Lin is a Ph.D. student in Information Management Department at National Yunlin University of Science and Technology, Taiwan. He received his master degree in Information Management from National Yunlin University of Science and Technology, Taiwan in 2002. His research areas are programming learning environments and ontology.

Yen-Hung Lin is a software developer at National Yunlin University of Science and Technology, Taiwan. He received his master degree in Information Management from National Yunlin University of Science and Technology, Taiwan in 2009. He is interested in developing open source software for on-line publishing and learning.