A web-based programming learning environment to support cognitive development

Wu-Yuin Hwang, Chin-Yu Wang, Gwo-Jen Hwang, Yueh-Min Huang, Susan Huang

Article info

A B S T R A C T

Web-based programming has become a popular and vital issue in recent years. The rapid growth of various applications not only demonstrates the importance of web-based programming, but also reveals the difficulty of training relevant skills. The difficulty is owing to the lack of facilities such as online coding, debugging and peer help to assist the students in promoting their cognitive development in web-based programming. To cope with these problems, in this paper, a web-based programming assisted system, "WPAS", is proposed, which is able to support five programming activities with various difficulty levels of cognition based on Bloom’s cognitive taxonomy. WPAS provides online coding, debugging and annotation tools to conduct the training and peer assessment for web-based programming. Experimental results of 47 undergraduate students show that the innovative approach is helpful to the students in improving their cognitive development in Web-based programming. In addition, according to the results of the questionnaire, most of the participants perceived the ease of use and usefulness of the proposed system. Therefore, this study suggests that teachers could design Web-based programming activities supported by the WPAS system to improve students’ cognitive development in web-based programming.

1. Introduction

In programming learning, continuous practice is required to ensure that the knowledge is retained (Truong et al., 2003). It has been found that actively and periodically scheduled learning is important for students to attain high levels of achievement (Hwang and Wang, 2004). In the Internet era, a Web-based environment provides a convenient way for language programming. In this circumstance, well-designed programming activities with assisting tools play an important role, and students’ cognitive development in programming should be taken into account in the designed activities (Lister and Leaney, 2003). However, most of the existing teaching methods for programming learning are inclined to place emphasis on students’ coding skills rather than on their cognitive development in language programming (Buck and Stucki, 2001; Lister and Leaney, 2003).

In programming courses, there are several critical issues to be considered, including the ways to stimulate students’ interaction in or after class, methods to enrich students’ learning experiences, and facilities to assist students in sharing knowledge with their classmates. Moreover, student learning is a process of interaction between a set of inner experiences of the learners and the environment (Slattery, 1995). In traditional programming courses, the problem solving based programming is considered to be a promising approach; furthermore, students are often asked to write complete programs to solve problems as soon as possible (Lister, 2001). Nevertheless, researchers have indicated that problem solving is in fact a necessary but not a sufficient criterion for programming (Winslow, 1996; Rist, 1995). The main difficulty faced by novices is expressing problem solutions as programs. Thus, the coverage of programming comprehension and how to apply programming comprehension to generate programs must remain an important focus. Recently, Robins et al. (2003) have proposed a programming framework for novices which highlights three programming procedures and the corresponding knowledge needed to complete each procedure: that is, the knowledge of planning methods to design a program, the knowledge of a programming language to generate a program, and the knowledge of debugging used to evaluate a program. Therefore, the cognitive development of programming to obtain the above knowledge needs to be considered in designing programming activities.

In traditional programming learning environments, such as computer classrooms, it is not easy to promote students’ cognitive developments in programming if learning activities and learning-assisted tools are not well integrated. For example, in addition to generating the whole program, program gap filling and peer program assessment play vital roles in building the programming cognitive development of students. Program gap filling can help students in developing their skills of solving sub-problems such as...
Bloom’s cognitive taxonomy (1956), this study attempts to design a series of web-based programming learning activities to enhance students’ cognitive developments in web-based programming. Moreover, a Web-based Programming Assisted System, WPAS, has been developed to support the learning activities. From some practical applications of WPAS, it can be seen that the innovative approach is helpful to the students in both the cognitive and affective aspects for learning Web-based programming. In detail, after conducting the experiment, the pedagogical findings show that the “program gap filling” activity is more difficult and needs much more consideration in the programming learning than “program debugging”. The activity of “peer assessment” is found to be the most strongly related to students’ learning achievements, and it is the best predictor variable of learning achievements. As for evaluating the WPAS, most of the students are satisfied with the system.

2. Literature review

In this section, the literature of Bloom’s taxonomy as well as its importance in terms of the design of programming learning activities is discussed; moreover, the TAM (Technology Acceptance Model) model (Davis, 1989), which is adopted in this study to evaluate how users come to accept and use the innovative approach, is also introduced.

2.1. Bloom’s taxonomy of educational objectives

Bloom (1956) proposed a classification framework for writing educational objectives, which has been adopted in many educational domains. The taxonomy consists of six educational objective levels (Bloom, 1956):

1. Knowledge: Knowledge is defined as the recalling of previously learned data or information. Knowledge represents the lowest level of learning outcomes in the cognitive domain.
2. Comprehension: Comprehension is defined as the real understanding of the meaning of a concept, which implies that one can interpret the concept with one’s own words.
3. Application: Application is defined as the ability to use a learned concept in a new situation.
4. Analysis: Analysis is defined as the ability to separate material or concepts into component parts so that its organizational structure may be understood.
5. Synthesis: Synthesis is defined as the ability to put parts together to form a new whole.
6. Evaluation: Evaluation is defined as the ability to judge the value of material for a given purpose.

According to Bloom, each level of cognitive development depends upon the behaviors and the knowledge which are acquired at the previous levels. In programming learning courses, students should be facilitated to develop their cognitive structure of programming in an appropriate way. In other words, learners’ cognitive development in programming should be taken into account during the design of programming learning activities. Lister and Leaney (2003) illustrated the following learning activities that students are expected to learn in programming courses:

1. Knowledge activities including “memorize”, “state”, “name” and “recognize”. For example, students memorize the elements, syntax, structure and methods of one programming language.
2. Comprehension activities including “restate” and “translate”. For example, students restate how a program executes.
3. Application activities including “calculate”, “write” and “solve”. For example, students accomplish a specific programming task such as giving partial code or gap filling according to some expressions.
4. Analysis activities including “categorize”, “differentiate” and “discriminate”. For example, students identify whether a complete program is correct or not.
5. Synthesis activities including “create”, “design” and “plan”. For example, students generate a complete program to solve a problem.
6. Evaluation activities including “assess”, “evaluate” and “judge”. For example, students review programs written by other students and give comments, then the instructor grades the comments.

Programming concepts cannot be directly transferred from instructors to students; they must be acquired actively by the students (Ben-Ari, 2001). This research applies several programming learning activities, based on Bloom’s taxonomy of cognition and Lister’s research, to facilitate students’ active learning and continuous practice in an experimental course.

2.2. Programming learning activities

Kolb’s Learning Styles Inventory (LSI) suggested a four-phase cycle of learning including concrete experience, reflective observation, abstract conceptualization, and active experimentation (Kolb, 1984). According to Kolb, concrete experience is a good starter during students’ learning processes. That is, in programming learning courses, practice is important for improving students’ learning. Students should be given enough practice opportunities in an environment where they can receive constructive and corrective feedback (Ben-Ari, 2001).

In programming learning courses; “coding to solve problems” is one of the most common learning activities for practicing programming. However, a single activity is not enough to help students build their cognitive development of programming step by step; that is, multiple activities in proper sequence, from simple to complicated, are needed for practicing programming. According to Affleck and Smith (1999), one of the main difficulties for novice programmers is accessing their prior knowledge and applying it to new problems. Using “fill in the gap” programming exercises is one of the ways to overcome the above problems (Liebman, 1986). The given code in these “gap filling” exercises is generally well known by students; the gap filling is a challenge which students should overcome by integrating their prior knowledge and the latest learned knowledge. Hence, “fill in the gap” programming exercises help students close the gap between existing and new knowledge (Van Merriënboer and Paas, 1990).

Also, computer programming classes often focus on teaching language syntaxes, analyzing problems and writing programs to solve problems. The class time has seldom been allocated for debugging practice. However, debugging training is much more important for novice programmers (Lee and Wu, 1999). Moreover,
Peer assessment has been applied in many different courses. Peer assessment is generally considered to be effective in promoting students’ higher cognitive skills (Fallows and Chandramohan, 2001), since students use their knowledge and skills to interpret, analyze and evaluate others’ work in order to clarify and correct it (Segers and Dochy, 2001; Ballantyne et al., 2002). The use of peer assessment is claimed to enhance students’ evaluative capacities (Fallows and Chandramohan, 2001).

2.3. Criteria for evaluating students’ programs

It is an important issue to evaluate and judge students’ programs in all kinds of programming courses (Jackson, 1996). The correctness of a program is a basic criterion when a teacher is judging the quality of students’ programs. To evaluate the correctness of a program, three kinds of programming errors are classified and employed: syntax errors, logical errors and runtime errors (Jackson, 1996). Syntax errors will occur when a programmer misuses the syntax of a programming language. Logical errors (or semantic errors) will occur when the logical thinking of a programmer is not correct enough. Runtime errors will occur in many situations. For example, if a programmer does not notice the value domain of a variable, an “overflow/underflow” runtime error could be encountered. The next important criterion to evaluate the quality of students’ programs is efficiency. The quality of programs can be finely distinguished by their efficiency when they run correctly. Many factors, like algorithms and data structures, will influence a program’s efficiency. The other important programming criterion is programming style (readability). A program with a good programming style is preferable since it is more readable and understandable.

When judging the quality of students’ programs, the above three criteria are taken into account in this study. Table 1 lists criteria for evaluating students’ programs from the previous related research (Jackson, 1996; Cheang et al., 2003; Sithiworachart and Joy, 2004).

2.4. Technology acceptance model: TAM

The TAM (Technology Acceptance Model) was developed by Davis (1989) to evaluate how users come to accept and use a technology. Based on user acceptance of the technology, TAM theory proposes perceived usefulness (PU) and perceived ease of use (PEOU) to explain a user’s attitude toward a system. As a result of TAM related research, PU was further employed to study a user’s intention to use the system, and it was found that PEOU also had a significant influence on the PU of a user who is currently using or learning an information technology. This research employs TAM theory to explore the “perceived usefulness” and “perceived ease of use” of the WPAS system.

3. Method

Based upon the above literature, a series of programming learning activities is proposed and conducted on the Web with the aim of improving students’ learning. The activities include programming concepts testing, program gap filling, program debugging, coding to solve problems and peer assessment, which correspond to the cognitive levels, evaluation, synthesis, analysis, application, knowledge and comprehension, in Bloom’s taxonomy.

3.1. Participants and subject

Forty-seven undergraduate students majoring in Management Information Systems participated in the experiment. Most of the students had basic computer concepts, but had no experience in computer programming. The experimental course, which was carried out from October 2005 to January 2006, was entitled “active server page (ASP) programming” and took place three hours per week in a computer classroom. In addition to the in-class practice, several learning activities were conducted in the WPAS environment as homework. With the help of the WPAS, the researchers were able to observe the learning behaviors of the students and analyze the data at the end of each activity to find some interesting learning phenomena.

3.2. Instructional design and learning activities

Three topics were arranged in the following sequence.

Topic 1: The programming concepts and syntax of ASP.
Topic 2: Web form, variables, data passing and objects of ASP.
Topic 3: Concepts of database programming and system implementation.

Due to the limitations of the experiment period, this research focused on topics 1 and 2 of the course. Fig. 1 illustrates the procedure of the five activities. The learning activities were conducted in the following sequence: programming concepts testing, program gap filling, and so on. As shown in Fig. 2, these activities correspond to one or more cognitive levels in Bloom’s taxonomy, that is, evaluation, synthesis, analysis, application, knowledge and comprehension. Note that knowledge and comprehension were considered together and corresponded to the first activity, programming concepts testing, in this research.

---

**Table 1**
The criteria to evaluate students’ programs

<table>
<thead>
<tr>
<th>Former studies</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jackson (1996)</td>
<td>Correctness, Style, Efficiency, Complexity, Test data coverage</td>
</tr>
<tr>
<td>Brenda, Andy, Andrew, &amp; Wee-Chong (2003)</td>
<td>Correctness, Efficiency, Maintainability</td>
</tr>
<tr>
<td>Sithiworachart and Joy (2004)</td>
<td>Correctness, Quality (comments, variable/function names, structure of the program, indent of the program)</td>
</tr>
</tbody>
</table>

---

**Fig. 1.** The procedure of the programming learning activities.
3.3. Design and development of the WPAS system

The WPAS system was designed and developed to support Web-based programming as mentioned above. The research data, such as the programs and the homework uploaded by the students, were collected and stored in the WPAS database. Moreover, students were encouraged to make some annotations on the learning materials (e.g., the demonstrated programs introduced by the teachers). The annotations and notes made by the students were helpful to them while they were reviewing the materials after class. Fig. 3 illustrates an overview of the WPAS system architecture. Three main tools were provided by the system:

3.3.1. Online coding tool

An online coding tool was utilized in the “program gap filling”, “program debugging” and “coding to solve problems” activities. For example, during the “program gap filling” activity, the students were asked to finish incomplete programs by writing codes in one assigned blank block. Once students finished the program gap filling and submitted their program within the online WPAS environment, the system would try to execute the program and instantly show the execution results with the source codes. If errors occurred, students could modify their source codes immediately online. Fig. 4 shows a snapshot of “program gap filling”. Also, with the online coding tool, students were allowed to upload programs for practice, and were encouraged to do so. After the programs were uploaded, they were depicted as a list of hyperlinks in WPAS. If the hyperlinks were clicked, the corresponding programs would be executed, and the execution results as well as the source codes were also depicted simultaneously, as shown in Fig. 5. This mechanism facilitated the students to modify and improve their web programs more conveniently and efficiently.

3.3.2. Annotation tool for assessing the programs

Students were encouraged to make annotations on the supplemental materials or programs uploaded by any other student. In this way, they could share their knowledge, comments and suggestions with each other. Fig. 6 illustrates annotations made by different students on a single program. The functionality of the annotation tool included highlighting, underlining and comment boxes (Hwang et al., 2007). Also, the annotation tool provided in WPAS was utilized in the “programming concepts testing” and “peer assessment” activities. During the “programming concepts testing” activity, students were asked to use the annotation tool to give their answers right beside the web-based assignments. As for “peer assessment”, students were asked to review the programs

WPAS

Fig. 3. Overview of the WPAS system architecture.
3.3.3. Searching for key words in source code

In WPAS, a keyword-searching functionality was also provided. Students could easily find related programs in the WPAS as needed.

3.4. The grading criteria of programming learning activities

The achievements of the programming learning activities were graded based on the following criteria or assessment approaches:

3.4.1. The grading criterion of “programming concepts testing”

A test that included open and closed questions was conducted, and was assessed manually by the teachers. The grading criterion of the test was simply the correctness of the answer to each question.

3.4.2. The grading criteria of “program gap filling”, “program debugging” and “coding to solve problems”

This research adapted three criteria, that is, “correctness”, “efficiency” and “programming style” (as described in Section 2.3), to evaluate the assignments for the “program gap filling”, “program debugging” and “coding to solve problems” learning activities. These criteria were announced before conducting each learning activity, so that the students could understand how their homework would be graded. Each student’s performance in the three activities showed their cognitive levels of programming in Bloom’s application, analysis and synthesis, respectively. The details of the three criteria are described as follows:

- **Correctness**: The teacher judged the correctness of each program by executing it and testing it. If a program was completely correct, 100 points was given for the correctness criterion. If the result was partially correct, 5 points was deducted per “bug”. On the other hand, if the program could not be executed due to syntax errors, zero points were given.

- **Efficiency**: In order to evaluate the efficiency of students’ programs, the time complexities of the programs were evaluated (Cormen et al., 2001). The time-complexity of a program is the number of steps that it takes to solve an instance of the problem as a function of the size of the input. If the time-complexity (a Big O notation) of one program for a problem submitted by a student was quicker than or equal to the time-complexity of the program provided by the teacher, then the student would get 100 points for the efficiency criterion; otherwise, a lower score was given.
Programming style: The students were taught and encouraged to write their programs using appropriate programming style, for example, writing program statements with clear structure, adding comments on each block of program codes and using library functions. The programs with better programming style were graded with higher scores.

3.4.3. The grading criteria of “peer assessment”

Five quality levels were employed to evaluate the peer assessment quality: “no comment”, “nonsense comments”, “rough comments”, “meaningful descriptions and explanations about the assigned program” and “meaningful suggestions about the assigned program”. Based on the quality of the five levels of comments, three teaching assistants reviewed all of the comments that were contributed by the students during the peer-assessment activity. Each student’s comments on the assigned programs were classified into one of the five levels, and were then graded with the corresponding score. The reliability of the scoring was investigated by Kendall coefficient of concordance. As shown in Table 2, the value of Kendall’s W is sufficiently concordant. The above criteria were employed to measure the quality of the occasions on which the students acted within each programming activity. And the score summation of all occasions had done by students in each activity represents the students’ achievement in each programming activity.

3.5. Research structure and research variables

The research structure is illustrated in Fig. 8. During the experiment, all students learned within the same learning period, were taught by the same teacher, and utilized the same WPAS system.
with the same content provided in the courses. That is, some variables were controlled. In Fig. 8, a bidirectional-dotted-arrow line shows the correlation between the two connected variables. The proposed learning activities were investigated to show their influence on programming learning achievement. The directional-solid-arrow line means regression analysis was used to get the prediction of learning achievement from the score of the five learning activities. The independent variables of this research include the scores for “programming concepts testing”, “program gap filling”, “program debugging”, “coding to solve problems” and “peer assessment”. The dependent variable is learning achievements. The score of “programming concepts testing” stands for the score earned by the students in the activity of “programming concepts testing”, and the score of “program gap filling” stands for the score earned by the students in the activity of “programming concepts testing”, etc. Moreover, learning achievements stands for the post-test score earned by the students in the final exam on completion of the course.

4. Results and discussion

4.1. Correlation between each programming learning activity

Pearson correlation was utilized to measure the relationships between students’ scores in each programming learning activity. As shown in Table 3, there exist significant correlations between any two pairs of students’ cognitive achievements in the programming learning activity. The highest coefficient (.608: between the coding to solve problems activity and the peer assessment activity) revealed that the ability of coding to solve problems was strongly associated with that of assessing programs. This finding was also explored deeply via the interviews. From the student interviews it was found that the “coding to solve problems” and “peer assessment” activities facilitated them in integrating their existing and new knowledge to solve problems, as well as to evaluate other programs, which both belong to the high level of cognition for learning programming.

4.2. The relationship among the scores of the five programming learning activities and learning achievements

Simple regression analysis was utilized to explore the correlation between the scores of the five programming learning activities and learning achievements. According to the results shown in Table 4, except for “concepts testing”, the scores of the “program gap filling”, “program debugging”, “coding to solve problems” and “peer assessment” activities were significantly positively correlated with the students’ learning achievements. Moreover, the obtained R value of “peer assessment” is .615, which is the highest. This reveals that “peer assessment” was the most critical activity related to learning achievements.

4.3. Multiple regressions between the scores of the five programming learning activities and learning achievements

Multiple regressions were used to further study the prediction of scores of the five programming learning activities on learning achievements. The results are shown in Tables 5 and 6.

In Table 5, the $R^2$ value ($R^2 = .466, F = 6.976, p = 0.000$) indicates the whole predictability is 46.6%, that is, the students’ score for the five programming learning activities can be used to predict 46.6% of their learning achievements. As the results of multiple regressions show in Table 6, the standardized coefficient $b$ of “peer assessments” was .457 ($t = 2.869$, $p = 0.007$) which was the highest of all. The results reveal that the students’ peer assessment activity score is the best for predicting their learning achievements. That is, the better the students performed in the peer assessment activity, the higher their learning achievements. Note that since the standardized coefficient $b$ of “programming concepts testing” is $-0.268$ in Table 6 and its $R^2$ square value is .279 in Table 5, we can not infer that “programming concepts testing” is of no help or has a negative impact on students’ learning achievements, because the correlation is positive between the score of the “programming concepts testing” and learning achievements. To
investigate whether the above five activities were sufficiently independent, collinearity was considered in multiple regressions. According to statistics theory, Vif (Variance inflation factors) measures how much the variance of the standardized regression coefficient $b$ is inflated by collinearity. According to Neter et al., there was no collinearity between the predictor variables if Vif value was less than 10 (Neter et al., 1985). In Table 6, all Vif values are between 1.581 and 2.151. This means that there is no collinearity between the score of the five programming learning activities. That is, each programming learning activity had its own effectiveness on learning achievements.

4.4. The results of independent sample $T$-tests: performances of high and low achievement groups in the five programming learning activities

The experimental class was further divided into a high achievement group and a low achievement group according to their post-test scores (i.e., their learning achievements). The students whose post-test score was in the top 27% belonged to the high achievement group, while those whose post-test score was in the bottom 27% were in the low achievement group. As shown in Table 7, the scores of the five programming learning activities of the high achievement group were all significantly higher than those of the low achievement group in each activity. That is, the better the students performed in each learning activity, the higher the learning achievements. Note that “program gap filling” ($p = .002$) is more significant than “program debugging” ($p = .023$), even though “program debugging” is supposed to be a higher cognition activity in programming learning. This finding was also supported by the values of $R$ square shown in Table 4 (the $R$ square value of “program gap filling” is larger than that of “program debugging”). This was an interesting phenomenon worth studying further by interviewing some students in both the high and low achievement groups after the experiment. From the interviews with the students in the high achievement group, it was found that the “program gap filling” activities, which asked the students to provide a block of program codes to work with the given codes, were harder than the “programming debugging” activities. Also for the students in the low achievement group, the “program gap filling” activities were quite difficult in comparison with the “program debugging” activities. Therefore, in arranging “program gap filling” activities, longer practice time and more support from the teachers are needed. On the other hand, the significance of an independent sample $t$-test in “program debugging” is .023. The significance is weaker than for the other activities, and the Standard deviation of the low achievement group was obviously high (SD = 28.01934). Two reasons were found to cause this result, namely, the number of “program debugging” assignments was insufficient and the “program debugging” assignments were comparatively easy for some students.

4.5. Analysis of the questionnaire

To investigate the students’ viewpoints about WPAS and the five programming learning activities, the students were asked to
fill out a questionnaire, which included three dimensions: perceived ease of use (PEOU) of WPAS, perceived usefulness (PU) of WPAS, and the PU of the five programming learning activities. The PEOU and PU were based on the Technology Acceptance Model (TAM).

4.5.1. Reliability and validity of the questionnaire

The questionnaires were given to 47 students in the experimental class and 46 valid answer sheets were obtained. The questionnaire consisted of 18 test items using a five-point Likert scale. The researchers utilized Cronbach’s $\alpha$ analysis to evaluate the internal consistency of each dimension of the questionnaire. As shown in Table 9, the Cronbach $\alpha$ values in all dimensions were higher than .80, and hence the questionnaire was considered to be highly reliable (Carmine and Zeller, 1979). Also, expert validity was used to evaluate the validity of the questionnaire. During the questionnaire design phase, all of the items were verified and validated by domain experts, and some ambiguous or unsuitable questions were modified and removed accordingly.

4.5.2. Results of questionnaire analysis

With respect to the perceived ease of use shown in Table 9, most of the students thought that the annotation tool and the online coding tool in WPAS were easy to use. They also indicated that the operations of the two tools could be learned quickly. With respect to the perceived usefulness listed in Table 10, the average ratings in items 8 and 10 were comparatively higher than for other items. Also, 15 students chose “strongly agree” for item 8. Thus, most of the students thought that the “online coding and execution” function provided in WPAS was useful for their programming learning. In item 9, 38 students (10 “strongly agree” and 28 “agree”) agreed that the online coding tool can increase their learning efficiency. Moreover, according to the results of item 11, the students thought it was helpful to learn Web-based programming with WPAS. Finally, most of the students were satisfied with the WPAS system overall (item 12). By summarizing the results of perceived ease of use and perceived usefulness, it can be seen that the students highly accepted the use of WPAS. As for the perceived usefulness of the five proposed activities shown in Table 11, most of the students thought that the four programming learning activities, other than “peer assessment”, could respectively improve their programming abilities (the average values of items 13, 14, 15 and 16 were all higher than 3.8). For “peer assessment”, one point is worth mentioning; that is, the average rating for item 17 was 3.50, which was the lowest in Table 11. It was found that 15 students chose “unable to answer” in that item, indicating that peer assessment was too difficult for some students to give meaningful suggestions concerning others’ programs. Thus, the teacher might need to give more support to those students who did not have high achievements in peer assessment. Finally, most of the students thought that, on the whole, the five programming learning activities could help promote their web-based programming concepts and abilities.

4.6. The results of the in-depth interviews

Eight students were interviewed by the researchers after conducting the experiment. Three students were from the high achievement group and the others from the low achievement group. These students gave some in-depth and interesting feedback and suggestions as follows:

(1) Regarding the five programming learning activities, the interviewees thought that the learning design was well-structured and ordered. Also, those few students who had previously taken programming courses thought that the learning activities in this course were more interesting than those in the other courses. For example, two students shared the same feeling:

“Compared with the programming course I had taken before, I am interested in the five programming learning activities in this class. I do not feel bored during the learning process. Various learning activities can train my thinking. They are helpful for me...”

(2) Students would like to have more practice opportunities immediately after lecturing in class. As some interviewees responded:

“I need more time to practice after lectures in class...”

“I think if the teacher can give us more time to do practice in class, we can learn better.........”

“During practice, I may encounter some problems if I don’t understand enough, then I can ask and get help from the teacher and classmates soon in class. This will improve learning a lot...”

(3) The program gap filling activity created more challenges than the program debugging. As four students in the low achievement group responded:

“The of program-gap-filling assignments were difficult. Even though I found several methods to solve the problem, I did not know which should be filled in the block...”

“I think the program-gap-filling activity is more difficult than the program-debugging activity...”

(4) Through the peer assessment activity, logical thinking will be stimulated, and programming skills will be enhanced. As one student in the low achievement group mentioned:

Table 7
The results of independent sample T-tests: average scores of high and low achievement groups for the five programming activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Group</th>
<th>N</th>
<th>Average score</th>
<th>Std. deviation</th>
<th>t</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming concepts testing</td>
<td>H</td>
<td>13</td>
<td>86.782</td>
<td>8.626</td>
<td>2.608</td>
<td>.017</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>13</td>
<td>74.625</td>
<td>14.423</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program gap filling</td>
<td>H</td>
<td>13</td>
<td>82.038</td>
<td>9.177</td>
<td>3.587</td>
<td>.002</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>13</td>
<td>70.692</td>
<td>6.768</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program debugging</td>
<td>H</td>
<td>13</td>
<td>89.653</td>
<td>7.738</td>
<td>2.562</td>
<td>.023</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>13</td>
<td>62.000</td>
<td>28.019</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding to solve problems</td>
<td>H</td>
<td>13</td>
<td>79.538</td>
<td>11.091</td>
<td>4.188</td>
<td>.000</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>13</td>
<td>52.807</td>
<td>20.165</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peer assessment</td>
<td>H</td>
<td>13</td>
<td>4.076</td>
<td>.734</td>
<td>8.654</td>
<td>.000</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>13</td>
<td>1.641</td>
<td>.700</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$t^*$ $p < 0.05$.

$t^*$ $p < 0.01$, H, high achievement group; L, low achievement group.

Table 8
Questionnaire dimension and the Cronbach $\alpha$ values

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Cronbach $\alpha$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceived ease of use of WPAS</td>
<td>0.8813</td>
</tr>
<tr>
<td>Perceived usefulness of WPAS</td>
<td>0.952</td>
</tr>
<tr>
<td>Perceived usefulness of the five programming learning activities</td>
<td>0.9088</td>
</tr>
<tr>
<td>Total Cronbach $\alpha$ value</td>
<td>0.9500</td>
</tr>
</tbody>
</table>
Although I often did not fully understand others’ programs in the peer assessment activity, I think that the activities will do me good."

Also, three students in the high achievement group gave the following feedback:

“I like to do peer-assessment……”

“I can review and check my programming concepts when I assess my classmates’ programs……”

“I suggest that the teacher provide more peer assessment activities for us……”

“Seeing and evaluating others’ programs can increase my programming experience……”

(5) The interviewee suggested that more “coding to solve problems” should be added. As two interviewees responded:

“I think the arranged time in the “coding to solve problems” activity should be extended, because it was really the opportunity for me to integrate my knowledge……”

“If the frequency of the “coding to solve problems” activity can be increased, I think my programming skills will be better……”

5. Conclusions and discussion

This study proposes an innovative approach for the cognitive development of programming learning. A Web-based Programming Assisted System (WPAS) that supports five programming learning activities from simple to complicated has been developed based on the approach. The motivation of this study was to provide appropriate activities and sufficient practice to students, and help them develop their cognition in programming learning. From the experimental results of a practical programming course, several interesting and important findings were derived. First, the students’ scores in each learning activity were positively related with the score in the next activity, which implies that a well-structured programming learning system can be developed. Second, students’ perceived usefulness of the five programming learning activities was high, which indicates that the system is effective in improving students’ programming learning. Finally, students’ perceived ease of use of the system was also high, which implies that the system is user-friendly and easy to use. These findings provide valuable insights for the development of future programming learning systems.
and ordered learning procedure which takes cognitive development into consideration is vital for programming learning. Therefore, teachers might need to pay more attention to the development of cognition in conducting programming learning activities. Through well-structured and well-ordered learning activities, students can easily build their knowledge step by step and will not feel frustrated during their programming learning process.

Second, the results of multiple regression analysis showed that the students’ peer assessment activity score was the best predictor variable for learning achievements. That is, peer assessment was the learning activity most strongly related to the students’ learning achievements. Therefore, in designing learning activities for programming training courses, peer assessment, a programming learning activity with high level cognition, needs to be taken into account more. Also, it would be better for the teachers to give more support to those students who have difficulty in doing such a high level activity.

Third, the questionnaire results regarding attitudes towards the value of peer assessment are fascinating as they directly contradict the performance data. That is, although some of the students earned a low score in the peer assessment activity, they thought peer assessment was useful for their programming learning. Indeed, with the attractive annotation tools provided by WPAS, students’ motivation could be enhanced because they can exchange programming skill and concepts through the comments made by annotation tools. Also, “peer assessment” included encouraging students to enter into discussion and a range of social variables which could account for these findings. With these tools, students would be facilitated to learn and help one another more outside of class, and this indirect effect could be responsible for improving their learning.

Finally, although well-designed learning activities and corresponding facilities such as WPAS are helpful to the students in developing the cognition of programming, the effectiveness of the WPAS in terms of students’ programming learning needs to be further investigated. A “pretest–posttest two-group quasi-experiment” will be carried out in the near future to perform such an evaluation. During the future experiment, a control group without WPAS and an experimental group with WPAS will participate. The effectiveness of the WPAS will be revealed in this future research. Also, baseline measures of programming knowledge will be taken to see if low achiever gains are greater or less than high achiever gains.

Acknowledgements

The authors acknowledge the anonymous reviewers whose suggestions and comments were helpful in the improvement of this paper.

References