K-040

Collaborative Learning Environment Based on COOP Model

Dinh Thi Dong Phuong, Fumiko Harada, Hideyuki Takada, and Hiromitsu Shimakawa
Computer Science Department, Ritsumeikan University, 1-1-1, Noji-Higashi, Kusatsu, Shiga, Japan
phuong@de.is.ritsumei.ac.jp

1. Introduction

At the present, programming training condition in many places is not good enough for students. Especially there are not enough teaching assistants – TA. In fact, there are usually 40 or more students practicing in a computer room but there is only one teacher or few TAs. This condition causes practicing time is not effective to both students and the teacher. Students cannot get guidance timely from teacher for their problems. They cannot progress their programming as their expectation. They are not satisfied and become bored with studying programming. Meanwhile, the teacher has to give guidance and answers exhaustively many questions from many students though these questions often include similar ones repeatedly asked or easy ones for other students to answer.

For the problems, it is necessary to build a collaborative learning environment to promote collaborative learning among students. Whenever a student gets in troubles, it consults with other students in its collaborative learning environment bravely to find out solutions. Since the students can get timely guidance, training quality is improved on the whole fashion.

The proposed collaborative learning environment, CoL-E, is composed of students whose co-learning is effective. The combination of students bases on each student programming ability and its group working one. Programming ability is measured by score of its source code. Group working ability is evaluated by its convincing opinions. They are opinions that can help others to solve certain problems.

2. Novice programmer education

Problems of novice programmer education can be classified as two main categories: One is that students do not master programming knowledge as well as experience from background, general to specific. Background problems include those related to understanding how computer works, what is a computer program and how to use tools. General problems include misunderstanding of programming concepts. Few experiences with programming processes such as analysis, design, coding, testing and maintenance are big obstacles. Specific problems are those that are associated with programming languages and particular programming matters such as usage of language constructs, expressions and strategies to apply them [1-3].

Another category is derivative problems from category one, and from insufficient programming training condition. When a student cannot make a successful source code, their programming confidence will be reduced. Long waiting time to be replied from teacher causes them to be bored with studying programming, which gradually reduces student motivation to study programming.

For novice programmers to be able to solve difficulties by themselves, a multistrategy error detection and discovery system - MEDD has been developed [4]. The system can enlarge the bug library time by time, and students can retrieve this library to solve their problems. The system is helpful to novice students, but there are considerable matters. The input of the system must be a program. This criterion is so difficult for the beginners. The bug libraries are based on patterns. Problems of novice are so wide range that they are hard to be classified into patterns. Retrieval from the bug libraries also takes much time. Understanding and practice the suggestion from the bug libraries are hard task for beginners.

Pair programming [5] is proposed as a solution for this training condition. For students who already master programming knowledge and enough experience, paired ones can solve more problems than single one. Their problem solving skills are increased. Their programs are of higher quality. Team working ability is also improved from pair programming. Especially 95% of the students enjoy and feel more confident with their programming after pair programming.

However, as educational view on novice programmers, if we let them pair to study programming, we cannot clarify exactly who have made a program because we have no means to manage all the pairs. To make the matter worse, if we valuate source codes, students good at programming would finish most parts of the work, leaving ones poor in programming idle. For the view point of education, this is far from the desired goal. Moreover, we do not know how to pair two students so that both of the two members can achieve most from their co-learning. We might combine a less experienced programmer with a more experienced programmer with the hope that the former will learn from the latter and can achieve the best result. However, the latter cannot reach to its highest achievement, because it has to spend time for the former.

3. Col-E Based on COOP Model

3.1 Co-learning

In an active, creative and cooperative class, students would seek for solutions from other students and other available information sources. They make up their decision, practice programming by themselves, and make the understanding from experience. This learning activity is called collaborative learning or cooperative learning, abbreviated as co-learning. We propose the Col-E as a co-learning environment.

3.2 Convincing Opinion

A convincing opinion - COOP is an opinion helpful or good for a solution on a certain problem. Suppose a student gets stuck in a problem during programming. It consults with others in its Col-E to find out solution. If a student offers COOP, the student would have high possibility to solve its problem. The student who has offered the opinion is also rewarded with COOP points from the remaining members who are in the same Col-E.

Generally, interaction of COOP points is really effective co-learning of students group. To propose an opinion on a problem, group members have to use from their knowledge and experience. The student having problem evaluates and tries these opinions. At the result, COOPs help the student get out of problems, and is a strong factor to promote co-learning among group members.

For each student, COOP points show both quality and quantity of its contribution to group co-learning. We call this contribution group working ability. It is a determining factor for effective co-learning of the students group. To encourage students co-learning, we can consider COOP points as important achievements as programming exercises scores.

3.3 Col-E Based on COOP Model

The Col-E is a group of students using a system based on the COOP model, as shown in figure 1. The combination of students is determined based on student programming ability and group working ability The number of each group should be 3 because of the balance of many matters. If there are more than 3 students in a group, one member would be interrupted too much while it has to focus on its own programming. The group member would not have a sense of responsibility to others, either. From the view point of the receiver, more than two different opinions are puzzling. Opinions from the other two members are enough to help the receiver. In case these opinions are not convincing, they can consult the teacher. A proper communication means must be stepped up among group members. Instead of face-to-face communication hard to be recorded, chat-based one through computer is supported with a proposed system.

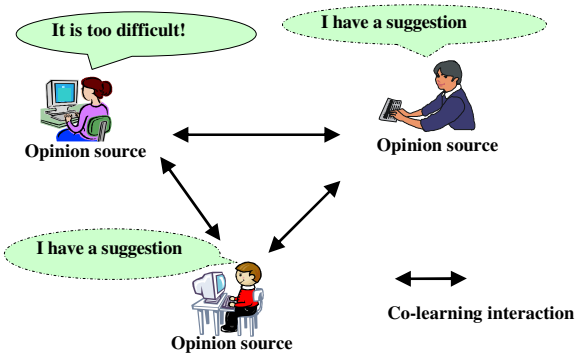

Figure 1. Co-learning environment

3.4 Grouping method

After each co-learning session, every student submits its source code to a teacher. The teacher grades these source codes for students. Each student will have two features as shown in figure 2:

(1) A score of its source code, and

(2) COOP points which are accumulated when it practices programming.

Based on these two features, all students will be classified into types. Figure 2 adopts 4 types: type I for strong programming ability and contribution, type II for strong ability but poor contribution, type III for poor ability but strong contribution, and type IV for poor ability and contribution.

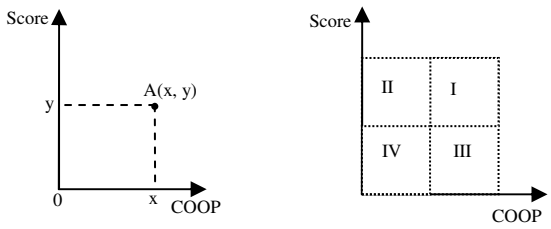

Figure 2. Student features and student types

For some preliminary sessions, students are grouped randomly. Let e be the number of the preliminary sessions. After session i ends, the following procedure is used to determine new student groups for session $i+1$, where $i > e$.

(1) Figure out type of each student.

(2) Evaluate whether a combination of students is good or not. If all scores from session i of all the group members are greater or equivalent to score of those of session $i-1$, the group is considered good. This good combination of student types is totaled up in a table.

(3) Group students based on their types and the statistics table.

3.5 Co-learning supporting system

It is a chat-based system with two main sub programs. Server program is to group students for effective co-learning. Client program is to collect students' source codes, communication and COOP points, as figure 3.

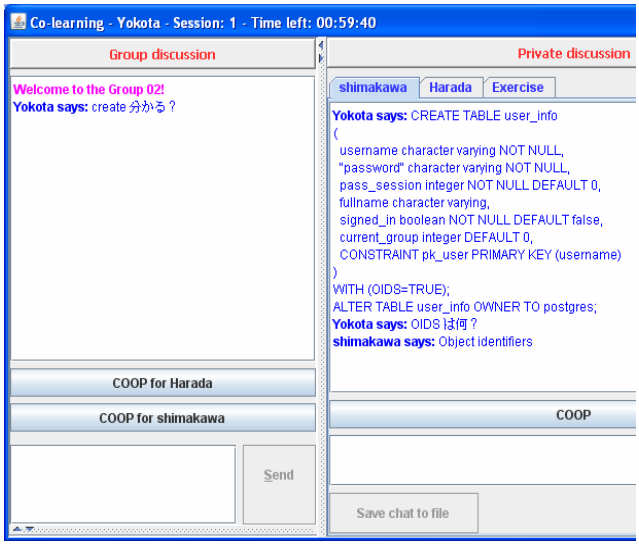

Figure 3. Co-learning client

4. Conclusion

The collaborative environment helps and encourages students co-learning during practicing programming. From there, programming learning and teaching are improved and promoted as the whole fashion.

The environment is best used for novice programmers in networked computer room.

References

[1]. A. Robins at al., Learning and Teaching Programming: a Review and Discussion, Computer Science Education, 13:2, 137-172, 2003

[2]. Garner, S., Haden, P., And Robins, My Program is Correct But It Doesn't Run: A Preliminary Investigation Of Novice Programmers' Problems, In Proceedings of the 7th Australasian Conference on Computing Education (ACE'05). 173–180, 2005

[3]. Brian Hanks, Problems Encountered by novice Pair Programmers, ACM Journal on Educational Resources in computing, Vol .7, No.4, Article 2, Jan 2008

[4]. RC Sison, M Numao, M Shimura, Multistrategy Discovery and Detection of Novice Programmer Errors, Machine Learning, 38, 157–180, Kluwer Academic Publishers, 2000

[5]. Laurie Ann Williams, THE COLLABORATIVE SOFTWARE PROCESS, Department of Computer Science, the University of Utah, 2000.