USING PAIR PROGRAMMING TECHNIQUES IN CLASSROOM ENVIRONMENT

Carlos Alberto Ynoguti[1]*, Afonso Celso Soares*[2]

Abstract *— For several years, software development companies have been using pair programming with great success. Pair programming is a technique that consists of two programmers working together on the same computer to develop a computer program: one of them, the driver, operates the computer, and the other, the observer, examines the work of the driver, looking for errors and thinking of possible alternatives that can improve the solution. This procedure is closely related to Collaborative Learning philosophy, extensively used in elementary and high school teaching, with very good results. Unfortunately, its use in undergraduate courses, especially in the engineering area, is almost absent. Therefore, we decided to adopt pair programming in a first semester computer-programming course, both in theoretical classes and in laboratory practical classes. This paper reports the results of a one-semester course using pair programming.*

Index Terms — pair programming, collaborative learning, computer science learning, extreme programming.

INTRODUCTION

Traditionally, students find introductory computer science courses very frustrating (in our institution, about 30% of them fail the subjects at each semester).

With pair programming learning, two students work simultaneously to solve a task (an algorithm or a computer program). In this technique, one of the students is the "driver" and has control on the pencil/mouse/keyboard and writes the algorithm or the program. The other, called the "observer", continuously and actively examines the work of the driver, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications of the work at hand. Examples of things noted by the observer are erroneous syntax, misspelling, and smaller logic mistakes, among others.

The student pairs apply a positive form of "pair-pressure" on each other, which has proven beneficial to the quality of their work products. At the end of the semester, the students were given a questionnaire about the pair-programming experience and most of them reported good impressions about this technique.

Also, this technique has proven to be beneficial for the teachers too. Some minor questions are answered inside the pairs. The number of exercises to correct is divided by a factor of two, enabling the teacher to give more exercises, and consequently, making a better evaluation of what issue is or is not being absorbed by the students. One important thing to note is that the number of cheating cases is reduced because collaboration is legitimized.

Cognitive theory can help explain why pair programming might be more effective than solo programming. In 1991 Nick Flor, a master's student of Cognitive Science at U.C. San Diego, reported on distributed cognition in a collaborative programming pair he studied. Flor recorded via video and audiotape the exchanges of two experienced programmers working together on a software maintenance task. In [3], he correlated specific verbal and non-verbal behaviors of the two programmers to known distributed cognition theories. One of these theories is "Searching Through Larger Spaces of Alternatives":

"A system with multiple actors possesses greater potential for the generation of more diverse plans for at least three reasons: (1) the actors bring different prior experiences to the task; (2) they may have different access to task relevant information; (3) they stand in different relationships to the problem by virtue of their functional roles. . . An important consequence of the attempt to share goals and plans is that when they are in conflict, the programmers must overtly negotiate a shared course of action. In doing so, they explore a larger number of alternatives than a single programmer alone might do. This reduces the chances of selecting a bad plan."

In this article, an experience involving pair programming learning technique in a first semester undergraduate computation course is presented. Advantages and disadvantages of this method are also presented and discussed.

Pair programming learning strategy is based on collaborative learning theory, which has been widely researched and advocated throughout the professional literature, mainly at the primary and secondary levels. For higher level courses, it's been adopted recently in some institutions with good results. This theory is further discussed in the next section.

[1] Carlos Alberto Ynoguti, INATEL – National Institute of Telecommunications, Av. João de Camargo, 510, 37540-000, Santa Rita do Sapucaí, MG, Brazil, ynoguti@inatel.br
[2] Afonso Celso Soares, INATEL – National Institute of Telecommunications, Av. João de Camargo, 510, 37540-000, Santa Rita do Sapucaí, MG, Brazil, acsoares@inatel.br .

Collaborative Learning

The term "collaborative learning" refers to an instruction method in which students at various performance levels work together in small groups toward a common goal. The students are responsible for one another's learning as well as for their own. Thus, the success of one student helps other students to be successful [5].

The collaborative learning points out the active participation and interaction, either between the students and the teacher or among the students.

Basic elements of collaborative learning

The basic elements of the collaborative learning method can be summarized into the following items[6]:

1. Group interdependency: the students, as a group, have a common goal and should work as an efficient team to reach it. The students are responsible for their own apprenticeship. This procedure helps in the apprenticeship of every member of the group.
2. Interaction: one of the goals of the collaborative learning is to develop the student's competence in working in groups. Each member of the group should accomplish his part of the task and spare some time to share his knowledge with his partner(s) and, on the other hand, receive the contributions of his partner(s).
3. Diverging thoughts: there shouldn't be a member that claims himself as the leader or the smart guy, but instead, a conscience that both of the members of the group can explain their own points of view, competence and perspectives. The activities should be created in order to demand collaboration instead of competition (complex tasks that require creativity and has several possible solutions).

Vygotsky's socio-cultural theory

The socio-cultural theory by Vigotsky about the learning process emphasizes that the human intelligence comes from our society and culture, and happens at first time because of the interaction with the social environment.

Another aspect of Vygotsky's theory is the idea that the potential for cognitive development is limited to a determined zone that he called "proximal development zone" (PDZ). He defines this concept as "the distance between the real development level, determined by the independent problem resolution, and the potential development level, determined by the problem resolution under a supervisor advising or in collaboration with more capable partners" [1].

It's important to consider that the PDZ varies with the culture, the society and the experience of each individual.

For a PDZ to be created, there should exist a joint activity that enables the interaction between teacher and students. The group work allows the confront and integration of different points of view, making the learning process richer and more interesting.

Of course, people learn by themselves naturally provided that there exist adequate and minimally stimulant contexts. However, if a teacher helps a student to analyze and reflect about his/her actions, the learning process is accelerated.

Experiments

Class characterization

Before describing the pair programming tests results, it's instructive to characterize the classes they were performed. The studies were carried out in an undergraduate, first semester computing class. In our institution, classes have typically 70-100 students. The course is divided into two parts: a theoretical part (60 hours) and a practical one (30 hours).

In the theoretical part, the students learn how to construct algorithms to solve simple problems, and use a pseudo-code language to construct these algorithms. All the algorithms are written with pencil and paper and the students are continuously invited to debug their algorithms simulating the behavior of the compilers.

Now, in the practical part, the students are divided into smaller groups (20 to 25 students) and use the Delphi® compiler to create their programs.

Grades are distributed in the following manner: 2 theoretical tests and 3 practical tests, taken individually. Also, during the semester, 10 to 15 exercises are assigned and corrected, and add a bonus up to 10 points in the final grade (grades range from 0 to 100 points). Part of these exercises was done individually, and part with a partner, using the pair-programming technique.

In other words, the pair-programming technique was used as a learning method, not as an evaluating one, although part of the grading had been obtained using this technique.

Solo and pair programming tests

As reported earlier in this article, the pair programming technique was used in the classroom and laboratory exercises. In the first part of the semester, the students developed their activities working alone. Because of the class size (74 students) and the class dynamics (teacher was always solving students' doubts during the exercises), cheating was difficult to control, and it was common to see 4, 5 or even 6 exercises with identical (and wrong) solutions. It was clear from these results that only a few students did the exercises. The others just cheated. It was a frustrating result, because it's clear from these facts that students were more interested in the bonus points than in learning how to program.

To try to change this behavior, the students were told that if the teacher found more than two exercises with identical solutions, the grade would be divided by the number of identical exercises. After this, the number of cheating cases dropped, and so did the grades.

Of course, as a result of this scenario, the students' grades in the first tests were not good.

In the second part of the semester, they were given an article about pair-programming [2] to read and had a brief explanation about the new working method. After this, they were invited to test on the new working methodology. None of them was obligated to work in pairs, but the teacher encouraged them to try the method before deciding how they would like to work. These activities were done both in classroom, with algorithm design problems, and in the laboratory environment, with code design problems.

Pairs were formed in a free way. No constraints were made in this sense and, in general, the students chose their friends or the students that was seated nearby, and later, it was noticed that this procedure was an error, as shown in the next section.

At the end of the semester, the students were invited to answer some questions concerning the pair programming experience. The questions were extracted from a work by Williams & Kessler [2] with some minor modifications. The answers the students gave will be commented later in this article.

Results

Cheating cases dropped dramatically, and the number of different solutions raised enormously. Also, the solutions varied from very sophisticated ones to very complicated and inefficient ones, but the great majority of the designs met the specifications of the problems, a result quite different from the first part of the semester.

The classes became very noisy, but this was because the students were really discussing solutions and alternatives to solve the problems and actively participating in the class. Also, when students asked the teacher to clear doubts, they came with more elaborated questions, not trivial ones, so the easy answering doubts were cleared independently by the students.

Specifically in the algorithms made in class, syntax errors are very common because of the lack of a compiler that reports them to the programmer. With pair programming this kind of error dropped dramatically.

Now in the laboratory environment, it was noted that the students waste less time doing other activities (such as talking, surfing on the internet, etc.) because of the partner's pressure. Also, they learn not only the theoretical aspects of programming, but also get some tips from their classmates: hot keys, typing tricks, help usage and other things were learned just by observing the partner working.

A final feature of this method is that, as students work in pairs the task of correcting the exercises is approximately divided by a factor of two, and then it can be possible to give more exercises during the semester and keep a closer look on how the students are assimilating each part of the subject and reinforce the weak areas.

Questionnaire analysis

The questions taken from [2], and the most common answers, were:

1) *It has been said among teachers, "You do not know it unless you can teach it." Do you find any value to yourself in explaining your work to your partner?*

Many students reported that when explaining some subject to his/her partner, they had to elaborate it in a more detailed fashion, so they learned a little bit more and noticed several aspects of the subject during this process.

2) *Do you feel you have learned anything just by reading your partner's code?*

The great majority of the students answered "no" to this question. They reported that they learned almost nothing by observing their partners working.

3) *What was the biggest hurdle you have had to overcome as a collaborative programmer?*

Accepting another strategy, different from what they had traced in the principle, was the most common problem reported.

4) *What kinds of things does the non-driver do as he/she observes?*

Some syntax errors, erroneous indentation, and minor logic mistakes were the most frequently answers found.

5) *What do you think is the biggest advantage of collaborative programming?*

Most of the students reported that the big advantage of pair-programming is that they could perform better algorithms, with fewer errors and in less time.

One of the students reported an interesting fact: during one development section, he didn't know how to solve the first part of the problem, and his partner helped him. In the second part, the opposite happened, he found the answer that his partner couldn't. So, working separately, both of them would fail the exercise, but working together, they could accomplish their goal.

6) *What do you think is the biggest problem with collaborative programming?*

The main problem the students found in pair-programming practice was when their partners didn't accept different ideas or suggestions. Some of the students also reported that their partner simply did nothing, not cooperating for the solution of the problem.

Drawbacks

As any other learning methodology, pair-programming has its drawbacks too. Analyzing the questionnaire answers and based on our own observations, we can list the following drawbacks in using pair-programming methodology in a classroom environment:

Some students really dislike working in pairs, and prefer to work alone. Some of them not even tried to work in pairs, preferring to work alone, even scoring only poor grades.

Pair choosing is another aspect that must be addressed carefully. They must be formed with one student that has a higher knowledge/skill level than the other, so he/she can help his/her partner. Pairs with two low knowledge/skill level students didn't worked also, because none of them could help each other. A student with very low knowledge/skill level together with a very high knowledge/skill level is another kind of pair that doesn't work, because the "expert" student quickly becomes bored with his partner and resolves the problem alone, without explaining the solution to his/her partner.

Another thing that must be taken into account for the pair programming to work properly is the personality of the partners. In our classes, we observed that students with a dominant profile tend to not accept suggestions and critics and try to resolve things by themselves. On the other hand, passive students tend to accept their partners solutions and avoid giving opinions, even when they notice something that is clearly wrong.

The last thing we observed in our classes is that there are some students that became addicted to pair programming and could not develop solo programming anymore. Probably this is the case of a passive student that agrees with everything even not understanding what the partner is doing.

Conclusions and Future Work

In this article we described the pair-programming method when used as a learning tool and showed the relationship between this method and the collaborative learning theory and the Vygotsky's socio-cultural theory.

In general good results were achieved, and most of the students were satisfied with their own performance in the course. We noticed problems in some pairs due to great difference in knowledge, when one of the partners was too dominant or too passive, and when partners had personal differences.

Also, some students could not maintain the performance of pair programming when working alone. This fact may indicate that not all the problems would be solved in pairs; students have to have some problems to solve alone to identify their strong and weak points, and have a more realistic view of themselves.

In our institution, a psychological profile evaluation is made for all the students by a specialist, and for the next semester, this information will be used to try to avoid problems with dominant/passive partners in the working pairs. Also, the knowledge/skill levels should be used to form the pairs, and the first test of the semester can be used for this purpose.

References

[1] VYGOTSKY, L. S. "A formação social da mente", *Martins Fontes*, São Paulo, 1994.

[2] Williams, Laurie A. & Kessler, Robert R. "Experimenting with industry's Pair Programming model in the computer science classroom", *Journal on Computer Science Education*, March 2001.

[3] Flor, N. V., & Hutchins, E. L. "Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance". *Paper presented at the Empirical Studies of Programmers: Fourth Workshop*, 1991.

[4] Williams, Laurie and Kessler, Robert R. "The Effects of Pair-Pressure and Pair-Learning on Software Engineering Education." *Conference of Software Engineering Education and Training*, 2000.

[5] Gokhale, Anuradha A. "Collaborative Learning Enhances Critical Thinking", *Journal of Technology Education.* Volume 7, Number 1 Fall 1995.(http://scholar.lib.vt.edu/ejournals/JTE/jte-v7n1/gokhale.jte-v7n1.html, (11/22/2002))

[6] http://www.minerva.uevora.pt/cscl.