Extreme Learning of Programming – A Methodology Based on eXtreme Programming for Programming Teaching-Learning

Eustáquio São José de Faria [1]
Keiji Yamanaka [2]
Josimeire do Amaral Tavares [3]

Faculdade de Engenharia Elétrica - Universidade Federal de Uberlândia, Uberlândia, Minas Gerais

[1] eustaquio@pucminas.br
[2] keiji@ufu.br
[3] josycbelo@gmail.com

Abstract. *The present work has been developed intending to propose the usage of collaborative practices in teaching programming in such disciplines. A methodology for programming teaching-learning named eXtreme Learning of Programming – XLP has been developed. The methodology is based on an agile methodology known as eXtreme Programming (XP) and on a Cognitive Programming Model. To justify the usage of this methodology, it is well known that the application of Pair Programming contributes to the increase of students' permanence in computer courses or alike due to motivation, sense of responsibility, and knowledge sharing provided by the social-cognitive conflict obtained from the pairings. Empirical researches are being done in an Information Systems course. Partial obtained results can be seen at the end of the paper.*

Keywords. *Cognitive Programming Model, eXtreme Learning of Programming, eXtreme Programming, Methodology, Pair Programming, Socio-Cognitive Conflict.*

(Received January 11, 2010 / Accepted July 28, 2010)

1. Introduction

Some graduate students, and even post-graduate ones, are not used to discipline and systematization needed for building programs. This is one of the factors responsible for the high rate of flunking in computing courses. Based on empirical data collected during the teaching of disciplines such as "Algorithms and Programming Techniques" and "Algorithms and Data Structures" between 2001 and 2007, it has been observed an over 35% flunking rate. Intending to identify the reasons and propose viable solutions, during discussions with other teachers, it has been observed that this rate is generally over 15% in disciplines such as "File Organization and Management" and "Programming Languages". These disciplines form the basic circle of computer programming and are the ones that most contribute for the high level of evasions observed in the first semesters in most computer courses.

By doing a bibliographical study on the area, it has been noted how long the problem lasts. According do Soloway [8], one of the problem's causes is related to the fact that most of programming books focuses on syntax and semantics of the languages, though such topics are not the biggest problem faced by newbies when programming. There is no doubt that the real problem faced by newbies is in "how to join all the pieces together" grouping and coordinating the program components. It is necessary much more than teaching them specific-language instructions; what it is really necessary is teaching newbies how to abstract a problem, its solutions and the acknowledgment of finding solutions to solve it.

The present work intends to explore the benefits of the strategy of working in teams. The pairing of developers, known as Pair Programming, is a practice that shows a great pedagogical potential in the teaching of computer programming techniques.

It has been proposed a methodology for programming teaching-learning based on the concepts of eXtreme Programming (more specifically, in Pair Programming) and in cognitive programming model developed by Lui and Chan [5]. The methodology has been nominated eXtreme Learning of Programming – XLP. The general goal of this research is to verify whether its usage contributes or not for a greater permanence of students in initial disciplines of computer programming in computer courses or alike.

The following sections describe (2) historical informations about Pair Programming; (3) the Lui and Chan's cognitive programming model; (4) the description of XLP methodology; (5) experiments in an Information Systems course; and (6) the conclusions and further works.

2. Pair Programming

Pair Programming (PP) is a collaborative practice of software development – which has been added to eXtreme Programming (XP) as one of its 12 key-practices – in which two developers work at the same time in a single computer and in the same programming task. One of the developers is commonly called Driver. The driver controls both the keyboard and mouse, and also does the programming task. The other, known as the Navigator, watches the driver's work and offers advice and ideas. The navigator constantly checks the entered data to identify tactical and strategic mistakes, while he or she looks for syntax and logical errors as well as implementations that disrespect pre-established rules imposed during the project. The developers switch roles at regular intervals.

PP is very promising to active students that learn by social interaction [10] and professionals that have abilities in collaborative work and communication – it occurs because PP is based on Piaget's theory on social-cognitive conflict. His theory describes that intelligence is not an individual property, it is a relational process between the person and other people that build and organize, together, their actions towards the physical and social environment (DOISE and MUGNY [3] *apud* GUERRERO [4]) – by such perspective, it is believed that conflicts or even questioning between the developers who participate in pairing sessions generates knowledge whatsoever.

Another interesting PP aspect is the continuous software revision. By continuously revising the project, codification and tests, the navigator guarantees the production of a better-quality system in relation to a similar project developed individually – it is important to point out that such revision is also a continuous source of conflict and questioning.

However, PP is not a new practice. Still in 1995, Constantine [1] has done one of the first reports in which has been observed the usage of paired developers in software development [6]. In the same year, Coplien [2] published his book about software production process suggesting an organizational standard of paired development.

These reports have made software development specialists curious. Motivated by this curiosity, Nosek [7] has published an experiment on PP and has concluded that such practice has increased 100% the developers' performance, and also has made the problem-solving process pleasurable. Nonetheless, Williams [9], after having applied a structured experimented in a Software Engineering classroom, in Utah University, has confirmed the reports of [7] and has shown that software development with PP results in a more reliable product due to a lesser bug rate.

Since then, several studies have been done to verify PP's benefits in the software development process and in the teaching-learning process of programming techniques in computer-related courses. The great consensus among the studies found in the literature involves the usage of PP as key-practice in the teaching of programming techniques. Even the authors of market-like works (researches done with professional developers) believe in the benefits PP aggregates to the teaching-learning process.

3. Lui and Chan's Cognitive Programming Model

According to section 2, many studies on Pair Programming have been found in the literature. Such papers have been unanimous on the pedagogical aspect of Pair Programming. However, the reported experiments have not described any structured model to articulate Pair Programming, except for Lui and Chan's work [5]. The authors have proposed a cognitive programming model called CPM (Cognitive Programming Model) in which developers should, during software development activities, identify the problem and develop a solution passing by six mental activities: (1) Definition; (2) Representation; (3) Model; (4) Schema; (5)

Algorithm; and (6) Code. It's important to point up that CPM also applies to individual programming. A description of each of the six mental activities can be seen below:

- Definition – When developers receive user requirements or programming tasks, it is expected that they can be able to recognize problem's basic perceptive elements, so they can understand it;
- Representation – Once the problem becomes understandable to developers, they must explore variables, functions, states, and all their inter-relationships provided by the problem, which can be used to represent and formalize it;
- Model – a representation merely describes the variables' and functions' states that are given by the problem. Unfortunately, many suppositions or events are not completely described. To completely model the system's behavior in a way that solves the problem, it might be necessary, inductively or deductively, the inference of unknown states in its description. It must be known the way in which the possible conditions (or the facts) are deducted or inducted;
- Schema –Scheming is the superficial structure that corresponds to a textual structure of a program, in order words, program explicit units, and the way they are arranged;
- Algorithm – Next level corresponds in getting deeper in the superficial structure. Algorithm is the description of the structure that corresponds to the representation of relations and indicates sequentially the logic and its control flow. Frequently, the logic is expressed in mathematical symbols, pseudo-commands (for instance, "read variable X") or structured language. It is important to point out that a solution for a determined problem is obtained during the building of the algorithm. This solution can be efficient, effective, and elegant or not.
- Code – The algorithm must eventually be expressed in computer language, so it can be executed. In a semiotic sense, it is expected that codification corresponds simply to the transformation of an algorithm's syntax to programming language syntax without semantics change. However, from a compiler's perspective, a program with correct syntax might not be executed due to hardware restrictions, for instance, lack of memory. Thus, coding can be very different from constructing algorithms. On such aspect, the most effective solution, or the most efficient one not always will correspond to the most elegant one.

The six mental activities are easily distinguished when developing solutions to complex problems. In simple problems, they are normally grouped and are hardly noticed. However, stimulating the distinction of these activities can lead students to comprehend the process of solving computer problems better.

4. e*Xtreme Learning of Programming* - XLP

Although many practices studied in the current section can also be applied to individual programming, it is believed that they obtain better results when they are used to promote collaborative programming learning.

XLP is based on the following theory: ***"When busy with a project and test planning activities, before the coding activity (followed by continuous revision), paired students produce better-quality software and promote knowledge share between each other"***. This means that students contribute to their partner's learning during paired project, coding and test activities and, as a result, produce better software. This theory is in agreement with the extreme programming theory, when defending the importance of knowledge sharing provided by Pair Programming.

It's believed that 7 out of 12 XP key-practices help the application of the XLP, which are: (1) planning; (2) simple project; (3) tests; (4) continuous project improvement; (5) Pair Programming; (6) collective ownership; and (7) standard code. According to these 7 XP key-practices and to the cognitive programming model proposed by Lui and Chan [5], the students should:

1. During planning practice: recognize basic perceptive elements of the problem to comprehend it ("definition"); describe and classify the current software's needed requirements; define the scope, explore variables, functions, states, and all their inter-relationships provided by the problem to represent it and formalize it ("representation"); estimate the work due date; determine the pairing process to be followed (who starts as pilot and who starts as a navigator, the elapsed time for each role switch, the preferred activities by each team member, etc.); elaborate a complete declaration of system's goals and determine the standard rules to be followed during project and coding activities;

2. During project practice: recognize the way in which the possible facts or states are deducted or inducted ("Model"); develop, based on problem's modeling, the use-case diagrams and their descriptions and the class diagrams; develop the activity diagrams ("Scheme"); and produce the algorithm in order to indicate the logic and its controlling flow sequentially ("Algorithm");
3. During test practice: build test cases for each use-case previously produced; produce decision tables ("Representation"); and a test table composed by the needed identified variables;
4. During Pair Programming practice: produce, based on algorithm and standardized rules previously defined, the program's source code ("Code"); revise his/her partner's work on each line added to the code; request role changes with his/her partner (between pilot and navigator); and try to find tactical and strategic errors in the code during the development;
5. During continuous software improvement practice: try to get to know or produce more elegant and/or efficient alternatives for problem-solving;
6. Comprehend the importance of software collective ownership, which involves the developers' acknowledgement about his/her responsibility for the production of a better-quality system and for the knowledge sharing with his/her partner. In this sense, it should not be allowed, for instance, thoughts like "my partner has chosen a bad strategy…" or "my partner does not know this instruction…", on the contrary, thoughts like "we have chosen a bad strategy…" or "we do not know this instruction…" must be encouraged;
7. Promote standard code practice, in the sense of: producing standard rules of project and code during planning practice; and continuously revise project, tests, Pair Programming and continuous project improvement practices.

5. PP in an Information Systems Course

The methodology adopted in this work is comprised from three different statistical approaches, but with a common purpose: to verify or not the effectiveness of Pair Programming technique for students of Information Systems course. The first approach consists in comparing the dropout rate in the discipline Algorithms and Programming Techniques (APT) when Pair Programming method is used and when the traditional method is adopted. The second approach describes global data collection of a sample of students who had attended this same discipline under the method of Pair Programming; and the last approach demonstrates the analysis of student development who had participated in an extra-class study group of Pair Programming.

1st Approach: to carry out the first approach, it was used the Pair Programming technique with students enrolled in the discipline APT during the Spring/2008 semester. After that, it was analyzed the history of previous groups (since year 2000) in this same discipline using data collected at the Academic Secretariat. It was chosen, as an analysis of variables, the number of dropouts in each semester the discipline had had. Dropouts were considered those who had dropped out of the discipline, those who had not taken the global and special evaluation, and the ones who had failed for not attending the classes. From the analysis of these behaviors, the ratios of dropouts were obtained from the beginning of the course. This ratio consists in dividing the amount of dropouts by the total amount of students registered for the discipline.

From the ratios of dropouts in each period, a parameter study on dynamic behavior was necessary. It was observed a satisfactory difference among the data of the first semester of academic year 2008 in relation to those analyzed school periods; however, the aim of the present work was to verify the statistical meaning of this difference.

To perform statistical calculations the following formulas were used:

1) Ratio of proportions:

$$\rho = \frac{x}{n}$$

Where:
X: Number of dropouts in the group
n: Total number of students in the group

2) Formula for calculating the weighted Ratio:

$$\hat{\rho} = \frac{(x1+x2)}{(n1+n2)}$$

Where:
x1: Number of dropouts in 2008 group
x2: Number of dropouts in comparative group
n1: Total number of students in 2008 group
n2: Total number of students in comparative group

3) Formula for hypothesis testing:

$$Z\,cal = \frac{(\rho 2 - \rho 1)}{\sqrt{\hat{\rho}(1-\hat{\rho})(\frac{1}{n1}+\frac{1}{n2})}}$$

Results found for calculation of inference statistics ($Z\,cal$) were compared with the value of normal approach equals to 1.65 (level of significance equals to 0, 10 → α = 0.1), obtained with a statistical band Table that was exactly proposed for inference problems in data statistics, so:

1. For -1.65 ≤ $Z\,cal$ ≤ 1.65, it is accepted H0 (with a hypothesis of 90% sure, the difference between the ratio of dropouts of 2 groups is not statistically significant).
2. For $Z\,cal$ < -1.65 and $Z\,cal$ > 1.65, it is rejected H0 and accepted H1, (with a hypothesis of 90% sure, the difference between the ratio of dropouts in two groups must be considered statistically significant).

2nd Approach: one of the goals was to evaluate the satisfaction and skills by using the pairing technique. A questionnaire was applied to students who attended the APT course in the Spring/2008 semester. With the application of a questionnaire, it was possible to analyze and verify student's satisfaction during the pairing sessions in lab classes (practical class-based programming language).

3rd Approach: the third approach aimed to analyze the development of students who participated in the extra-class course of Pair Programming. A study group was created and many students of all grades were registered – making the sum of 30 students. The students attending this course were submitted to XLP methodology for four hours a week, being two hours on Fridays and two hours on Saturdays. Students were required to develop programs and solve some logic exercises based on It was requested that the participants developed programs and solved some logic exercises based on dynamics directed toward pair work.

The students were trained, since the first meeting, in collaborative skills. Rotation among pairs was encouraged with the purpose to find out which partners were more compatible or identified better with each other - producing activities more efficiently.

5.1 The Experiment

As aforementioned, for the first approach development, it was used data from the Academic Secretariat that can be observed in Table 1.

Table1: Students History in APT Disciplines.

Class-Semester-Year	Students Quantity	Dropouts Quantity	Dropout Ratio
1—1—2000	58	8	0.138
1—2—2000	65	12	0.19
1—1—2001	37	7	0.19
2—1—2001	40	9	0.23
1—2—2001	37	7	0.19
2—2—2001	39	15	0.39
1—1—2002	40	12	0.30
2—1—2002	43	6	0.1395
1—2—2002	26	6	0.23
2—2—2002	35	5	0.143
1—1—2003	41	7	0.17
1—2—2003	46	20	0.44
1—1—2004	61	30	0.49
1—2—2004	44	18	0.41
1—1—2005	50	9	0.18
1—2—2005	28	9	0.32
1—1—2006	54	8	0.15
1—2—2006	35	10	0.29
1—1—2007	33	20	0.61
1—1—2008	**50**	**7**	**0.14**

1st Approach: as previously mentioned, it was defined as nullity hypothesis (H0) the equality between the dropout ratio in the discipline Algorithms and Programming Techniques in the 2008 group and in other groups.

Similarly, the alternative hypothesis (H1) considers the difference among all the ratios which were compared in each test. The ratio of 2008 dropouts is constant, once it was compared to those ratios found in other researches.

It was defined for all comparisons that:

$$\mathbf{H_0: \rho 1 = \rho 2}$$

$$\mathbf{H_1: \rho 1 \neq \rho 2}$$

Where **ρ1** always corresponds to the ratio of 2008 dropouts and **ρ2** corresponds to the ratio of dropouts of group who is being compared to.

Only the first test (carried out with 2007 group) is described below. The results of other tests can be found in Table 2.

1° Test: 2008 Group versus 2007 group:
ρ1: Ratio of dropouts in 2008 group = 0.14
ρ2: Ratio of dropouts in 2007 group = 0.61
x1: Number of dropouts in 2008 group = 7
x2: Number of dropouts in 2007 group = 20

n1: Total number of students in 2008 group = 50
n2: Total number of students in 2007 group = 33

According to the formula for the weighted ratio:

$$\hat{\rho} = 0{,}33$$

Solving the Formula for the hypothesis testing, the following results were obtained:

$$Z\,cal = 4.48$$

Conclusion to the 1st test:

As 4, 48 > 1.65, the hypothesis is rejected with 90% sure that the dropouts of population ratios are similar.

Still, remaking the comparison with a level of significance of 0.05 and normal approach of 1.96, which corresponds to 95% sure, it is concluded that the population ratios of dropouts are not similar, once that 4.48 > 1.96.

Using a level of significance of 0.01, still, equity is rejected between the hypotheses. That is, with 99% sure we can conclude that the population ratios of dropouts are not similar.

Results have demonstrated that the difference is statistically significant as 90% as 95% and 99% sure, what is considered very positive for the research.

Table2: Hypothesis Tests Results.

Tests	Compared Classes	Students Number	Dropouts ratio	Zcal	Conclusions
1	1—1—2008 vs. 1—1—2007	50	0.14	**4.48**	STATISTICALLY SIGNIFICANT
		33	0.61		
2	1—1—2008 vs. 1—2—2006	50	0.14	**1.71**	STATISTICALLY SIGNIFICANT
		35	0.29		
3	1—1—2008 vs. 1—1—2006	50	0.14	**0.147**	NOT STATISTICALLY SIGNIFICANT
		54	0.15		
4	1—1—2008 vs. 1—2—2005	50	0.14	**1.875**	STATISTICALLY SIGNIFICANT
		28	0.32		
5	1—1—2008 vs. 1—1—2005	50	0.14	**0.55**	NOT STATISTICALLY SIGNIFICANT
		50	0.18		
6	1—1—2008 vs. 1—2—2004	50	0.14	**2.94**	STATISTICALLY SIGNIFICANT
		44	0.41		
7	1—1—2008 vs. 1—1—2004	50	0.14	**3.89**	STATISTICALLY SIGNIFICANT
		61	0.49		
8	1—1—2008 vs. 1—2—2003	50	0.14	**3.15**	STATISTICALLY SIGNIFICANT
		46	0.44		
9	1—1—2008 vs. 1—1—2003	50	0.14	**0.40**	NOT STATISTICALLY SIGNIFICANT
		41	0.17		
10	1—1—2008 vs. 2—2—2002	50	0.14	**0.04**	NOT STATISTICALLY SIGNIFICANT
		35	0.143		
11	1—1—2008 vs. 1—2—2002	50	0.14	**0.99**	NOT STATISTICALLY SIGNIFICANT
		26	0.23		
12	1—1—2008 vs. 2—1—2002	50	0.14	**-0.0069**	NOT STATISTICALLY SIGNIFICANT
		43	0.1395		
13	1—1—2008 vs. 1—1—2002	50	0.14	**1.86**	STATISTICALLY SIGNIFICANT
		40	0.30		
14	1—1—2008 vs. 2—2—2001	50	0.14	**2.69**	STATISTICALLY SIGNIFICANT
		39	0.39		
15	1—1—2008 vs. 1—2—2001	50	0.14	**0.63**	NOT STATISTICALLY SIGNIFICANT
		37	0.19		
16	1—1—2008 vs. 2—1—2001	50	0.14	**1.11**	NOT STATISTICALLY SIGNIFICANT
		40	0.23		
17	1—1—2008 vs. 1—1—2001	50	0.14	**0.625**	NOT STATISTICALLY SIGNIFICANT
		37	0.19		
18	1—1—2008 vs. 1—2—2000	50	0.14	**0.70**	NOT STATISTICALLY SIGNIFICANT
		65	0.19		
19	1—1—2008 vs. 1—1—2000	50	0.14	**-0.03**	NOT STATISTICALLY SIGNIFICANT
		58	0.138		

Among tests that were carried out, 8 (eight) rejected the equality among the dropout ratios. And 5 (five) of these 8 (eight) tests also rejected, 95% sure, the equality among the ratios. On the other hand, 11 tests demonstrated that the ratios are similar, once they did not reject the equality among them, representing for **H0:ρ1=ρ 2**.

2nd Approach: a questionnaire was applied to students for developing the 2nd approach. It consisted in giving this questionnaire to students who attended the APT discipline and were submitted to the Pair Programming technique in the Spring/2008 semester. Students were asked to answer the questionnaire about the difficulty they had had in computer programming. Considering all the interviewees, 91% said to have difficulty in computer programming, as it can be observed in Figure 1.

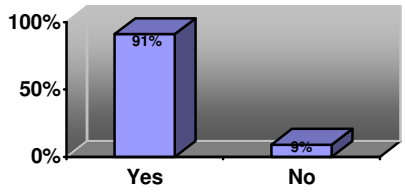

Figure1: Students' Difficulties in Programming

Students' evaluation concerning Pair Programming technique was analyzed. Under this aspect, 44% of interviewees considered it an excellent technique, while other 56% said the technique was good. Besides the "excellent and good options", there were the "regular and poor options", but none of the interviewees chose these two last options. This data can be detailed observed in Figure 2.

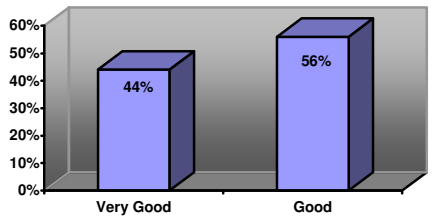

Figure2: Pair Programming Evaluation

Among the PP most noticed advantages by students, it can be distinguished at first place: "a better strategy for developing programs due to partner help" (80% of students claimed it as an advantage). In second place we can highlight: "Information sharing" (55% mentioned about that), and in third place: "interaction between the pair members" (cited by 30% of interviewers). They also cited: "better confidence in programming" and "small basic code development".

Interviewees were also questioned about the observed disadvantages. The largest representation was: "the partner can not collaborate with the development". However, this disadvantage can be minimized by the pair- pressure when the partner demands an active attitude from his partner.

Students were searched about the influence of Pair Programming in computer programming education. Among the interviewees, 94% affirmed that the technique has some influence on computer programming learning. Only 6% answered negatively, as can be observed in Figure 3.

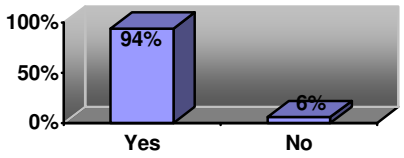

Figure3: Influence of Pair Programming in the Programming Learning

Considering this question, it was also verified that the Pair Programming technique has assisted the learning process of APT (20% of interviewees answered that it did not help, while 70% of interviewees said that it did, the technique aided them in their learning, and other 10% did not express their opinions). Some students still justified their negative question by saying that they had had little contact with this technique. Others, in turn, when they answered yes, they emphasized their experience through statements as the ones that follow:

- ✓ "The sharing of questions and answers develops programming skills."
- ✓ "Questions to be cleared by the teacher were discussed in pairs, what has generated a lot of knowledge about the topic."
- ✓ "Because we exchanged information, we had many doubts cleared which made computer learning easier."
- ✓ "I have learned a lot from my partner. By discussing with him, I figured out how a determined algorhytm

was done improving my performance in programming".

Interviewees were asked if they considered the technique in teaching computer programming interesting as a discipline. Only 9% of students answered negatively, while 91% of interviewees answered positively (Figure 4).

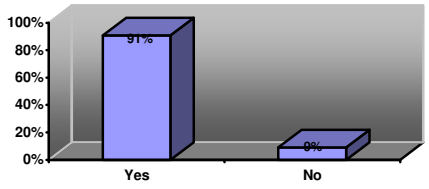

Figure4: Interest in Teaching Programming by the Use of Pair Programming Technique

According to the data collected, it was verified that students liked the experience of learning by working in pairs, and most of them believe that the application of Pair Programming technique a useful tool for learning computer programming.

3rd Approach: This stage of the work aims to analyze, in a perceptive way, the behavior and development of students who attended the extra-class course on Pair Programming. So, instructors observed the student's behavior during the classes, as well as their attitude, relationship with their partner, and wiliness to work in pairs. Therefore, it can be highlighted an evolution towards interaction between the partners. At the beginning of the paring process, the students demonstrated a shy behavior that was minimized along the term.

It was realized that some students had difficulties in keeping their position (navigator or pilot), so when they identified errors, "they tried" to correct them immediately, possessing the keyboard, even when they acted as the navigator.

It was also elaborated a questionnaire which was filled by the students during an extra-class course of Pair Programming. This questionnaire aimed to identify students' level of satisfaction due to pairings as well as their own development. Among the asked questions we can highlight: "Do you like programming, yes or no?" Table 3 below shows the answers obtained along the course.

Table3: Number of students who like programming

Research	Do not like programming	Like programming
1st (Week)	22	8
2nd (Week)	21	9
3rd (Week)	21	9
4th (Week)	17	13
5th (Week)	17	13
6th (Week)	17	13
7th (Week)	14	16
8th (Week)	14	16

We can observe in Table 3 that the number of students who like programming increased in 100% between the first and the last week of the course. The result is very expressive for our research because it shows that students, when paired, acquired a more positive attitude in relation to programming. It can still be observed students' satisfaction when they could solve, together with their partners, the proposed problems.

During the course, reasoning and logic activities were performed before the beginning of computer programming discipline. It was observed that students suggest that Pair Programming technique stimulates the reasoning because they have to think and find solutions with their partners before the writing of the code source, and even during the codification they intervene in the pilot's work suggesting and searching for explanations. These interventions are strong characteristics of Pair Programming, once there is great information exchange during the whole process.

Pair Programming was well accepted by the participants of extra-class course, as it can be observed in the transcriptions of student's statements below:

- ✓ "…sometimes I feel myself more confident when my ideas are accepted…"
- ✓ "… At some moments, we had to refrain ourselves, while navigators, not to intervene in the partner's task… it occurred for anxiety reasons…"
- ✓ "… It is necessary that the navigator gives the most useful information to the pilot. Any misplaced piece can take long to be fixed and, sometimes, this part can be influencing another one, causing a restart of the whole process…"
- ✓ "… Pair Programming has helped students solving simple problems. The pairs share their knowledge achieving a productive result, besides increasing each student's performance and will. Pairing helps to better developers' spirits, especially those who tend to give up when programming difficulties arise."
- ✓ "… There are moments, when we are programming, we focus only on running the program, while in Pair

Programming we try to program in the best possible way, optimizing commands and making their maintenance simple…"

6. Conclusions and Further Works

It can be observed, from data collected by the application of questionnaires, that students developed a favorable opinion in relation to Pair Programming technique for learning computer programming. This can be seen by the large amount of students (91%) who considered interesting the application of this technique in the discipline APT.

It was also observed important results in statistics through the application of hypothesis tests. So, the dropouts ratios obtained in the discipline APT during the previous years were compared to the dropout rate in the 2008 group (which worked under Pair Programming technique). A total of 19 tests were carried out. Among those tests, 8 rejected, 90% sure, the equality of dropout ratios. And the most interesting fact is that in all those 8 tests the dropout rate was inferior in the 2008 group.

This demonstrates that the experiment with pairs had a positive result. Also, it is believed that it collaborated for student's permanence in the discipline. However, it would be too early to state that Pair Programming was the only variable or the most important one that contributed for evasion reduction in that group. It is important to highlight that several other variables can have influence on such behavior, for example: (1) the 2008 group might have been formed by a very large amount of students who are retaking the discipline if compared to those previous groups. Thus, it is usually expected that the group who is retaking the discipline learns more easily than those who are seeing the subject for the very first time; (2) the 2008 group might have been formed by a lesser amount of students with financial problems than the other groups - certainly, financial matters can influence the number of evasion in a discipline or even a course, among other variables.

Therefore, it is necessary to perform a new research, as well as the application of effective technique in all the other computer programming disciplines in order to conclude effectively regarding the benefits that can be noticed through the Pair Programming.

As for tests that had not been so favorable, in which it was evidenced that the ratios of dropouts are equal, what did demonstrate not to have a statistically significant difference between the traditional method of computer teaching and the method applied in 2008 group, it is important to highlight that only two groups showed inferior results of evasion compared to the results of 2008 group.

As for extra-class study group, we have observed that students had a good development in relation to their satisfaction with programming activity.

It is important to highlight, from an experiment carried out during the development of this work that some benefits of Pair Programming are latent and very relevant, among them we can point out: (1) interaction between pair members; (2) knowledge sharing; (3) larger ease in developing programs due to partner help; and (4) better confidence in developing programs.

It is suggested, as further work, the application of the very same research in Computer Science courses offered by public higher education institutions in order to measure the benefits of Pair Programming in an environment where variables that influence the teaching-learning process are very different from private institutions.

It is also suggested a survey with students and former students in order to find out the possible reasons that led them to drop out the discipline Algorithms and Programming Techniques (APT) during all the periods covered by this research.

References

[1] Constantine, L.L. "Constantine on Peopleware". *Yourdon Press Computing Series*, ed. E. Yourdon, Englewood Cliffs, NJ: Yourdon Press, 1995.

[2] Coplien, J. O. "A Development Process Generative Pattern Language". *In Pattern Languages of Program Design*, J. O. Coplien and D. C. Schmidt, Ed. Reading Mass: Addison-Wesley, 1995, pp.183-237.

[3] Doise, W.; e Mugny G. "Socio-Cognitive Conflict and Structure of Individual and Collective Performance". European Journal of Social Psychology, vol 8, 1978, pp.181-192.

[4] Guerrero, P. V. T. "Interação Social: A Dominância em Situação de Aprendizagem". Dissertação de Mestrado. Universidade Estadual de Campinas (UNICAMP), Campinas, SP, 1998.

[5] Lui, M. M.; e Chan, K. C. C. “A Cognitive Model for Solo Programming and Pair Programming”. Proceedings of the Third IEEE International Conference on Cognitive Informatics, 2004, pp.94-102.

[6] Nawrocki, J.; e Wojciechowski, A. “Experimental Evaluation of Pair Programming”, Presented at European Software Control and Metrics, London, England, 2001.

[7] Nosek, J.T. “The Case for Collaborative Programming”. Communications of the ACM, Vol. 41, Issue 3, 1998, pp.105-108.

[8] Soloway, E. “Learning to Program = Learning to Construct Mechanisms and Explanations”. Communications of the ACM, Vol. 29, Issue 9, 1986, pp.850-858.

[9] Williams, L.; Kessler, R. R.; Cunningham, e W.; Jeffries, R. “Strengthening the Case for Pair Programming”. IEEE Software, Vol. 17, Issue 4, Jul/Aug, 2000, pp.19-25.

[10] Zualkernan, I. A. “Using Soloman-Felder Learning Style Index to Evaluate Pedagogical Resources for Introductory Programming Classes”. Proceedings of the 29th International Conference on Software Engineering, Minneapolis, Minnesota. 20-26 May, 2007, pp.723-726.