Student perceptions on the effectiveness of collaborative problem-based learning using online pair programming tools

MAIRA KOTSOVOULOU
Information Technology Department
The American College of Greece
Gravias 6, Agia Paraskevi
GREECE

VASSILIA STEFANOU
Management Information Systems
The American College of Greece
Gravias 6, Agia Paraskevi
GREECE

Abstract: - This paper describes a qualitative study of how undergraduate students majoring in Information Technology perceive the effectiveness and evaluate the learning experience of pair-programming. The phenomenographic research approach was used to analyze student interviews and revealed 4 categories of descriptions: Effective Problem Solving, Participation, Enjoyment and Coding. Pair-programming as a teaching methodology was commonly perceived as a positive experience. The resulting outcome space maps a logical hierarchy of students' conceptions of reality (categories of description). Findings of this research identify the factors that affect student engagement in a problem-solving process and can be used as a guiding principle on how to improve students' learning experience of computer programming.

Key-Words: pair-programming; problem-based learning; qualitative; phenomenography; information technology education; collaboration

1 Introduction

Traditionally, projects in undergraduate computer programming courses require students either to work exclusively on an individual basis or to take on smaller independent tasks as members of a team. There are a number of cases where students although are given an individual programming task to solve in class, they tend to ask their peers for "how-to" and "where-to-start" clues or to ask help in their debugging process. When assigned group projects most of my students prefer to work together rather than to split their project on smaller parts. Pair work seems to increase student level of confidence especially in introductory programming courses. The above observations gave me the incentive to initiate this study with an emphasis on peer synergy, distributed pair programming and collaborative learning.

Collaborative and cooperative learning concepts have been used almost interchangeably in most computer related studies. Cooperative learning has been defined as a collection of teaching techniques in which students work small groups to complete learning activities. Each student is responsible for his/her own learning but is assessed based on the group's performance. (Miller & Peterson 1980; Slavin & Madden 1989; D. W. Johnson et al. 2000)

McConnell (2000, p.26) summarizes the benefits of cooperative learning as:

a) helping to clarify ideas and concepts through discussion,

b) developing critical thinking,

c) providing opportunities for learners to share information and ideas,

d) developing communication skills,

e) providing a context where the learners can take control of their own learning in a social context,

f) providing validation of individuals' ideas and ways of thinking through conversation (verbalising),

g) offering multiple perspectives (cognitive restructuring), and

h) promoting constructive argument (conceptual conflict resolution).

Collaborative learning on the other hand is more of a personal philosophy rather than a set of activities and predefined structures. According to Bruffee *"Collaborative learning provides the kind of social context, the kind of community, in which normal discourse occurs: a community of knowledgeable peers"* (Bruffee 1984).

Both concepts are based on the constructivist theory where learning takes place through social interactions but there are many conceptual differences between cooperative and collaborative learning based on purpose, structure, student-teacher relationships and prescriptiveness of activities. Cooperative learning activities are highly structured and prescribed by the teacher. Each student is accountable to the group and to self.

Assessment is based on group work. Collaborative learning activities are not closely prescribed and group roles and structure is not strict. Students engage with "more capable others" who provide assistance. Emphasis is on learning rather on completion of a specific project. (Oxford & Dean 1997; Matthews et al. 1995; Panitz 1997).

Although the cooperative method seems more appropriate to characterize pair programming, most researchers refer to pair programming as a collaborative activity. In the context of this study on-line editor-sharing collaboration tools are used to enable distributed pair programming. Despite the conceptual and philosophical differences between collaborative and cooperative learning, this paper focuses on "working together" to accomplish a "common project" by using collaborative on-line tools.

Collaborative programming is not a new term. In the book "The Psychology of Programming", Weinberg (1971) explored the collaborative view of programming. Pair programming is a programming style in which two programmers physically work side-by-side, using the same computer to work on a common computer program. The process involves all the typical stages of software development, ranging from analysis and design to coding and testing, but differs radically in the way the process is actually implemented. In pair programming, one person is the designated "driver" and has control of the keyboard, the mouse and the pencil (for the analysis and the design phase). The "driver" is tasked with typing the actual code, drawing the actual designs, and generally executing hands-on tasks. The second person, the "navigator", continuously observes the work. The "navigator" has a number or tasks to accomplish. One of them is to monitor the tasks at-hand, as well as the overall process, and actively watch for problems (syntax errors, flaws in logic, omissions, etc). Other important tasks include considering alternatives, iteratively evaluating design and implementation decisions and looking up additional resources on a per-need basis. In the pair programming paradigm, it is very important that the two partners communicate on a constant basis, while exposing and explaining entire thinking processes to each other. (Williams 1999; Nosek 1998; Williams et al. 2000).

Kent Beck, the primary developer of the idea of eXtreme Programming, employs pair programming in his software development methodology. After years of research on agile programming methodologies, Beck has reached the conclusion that two programmers working together are more productive than two programmers working alone and produce twice as many solutions to a problem. This affords them a greater pool of solutions from which they can choose the best one and thus create a higher quality product with less bugs. (Beck 1999; Fowler et al. 2001)

Beck also stresses the importance of pairs rotating their roles at regular intervals, because each role assumes a different level of abstraction. (Beck 1999). This allows both pair members to bring their individual skillsets and levels of insight to each discrete role.

In pair programming, both members of the team collaborate constantly to create a common solution and produce a working computer program. In the process one help the other with the thinking, constructing and debugging procedure. The result of this common effort is a teamwork product upon which they are mutually evaluated.

Past research (both anecdotal by professional practitioners and empirical) has shown that two programmers sharing the same computer and working collaboratively on the same design, algorithm, code, or test can perform substantially better than the two working alone as far as speed, functionality and quality are concerned.

Empirical studies on the effectiveness of pair-programming in undergraduate level programming courses have shown that students are more confident, they enjoy programming more and ultimately produce higher quality code. (McDowell et al. 2003; Slaten et al. 2005; Williams et al. 2000)

The pair programming paradigm can be extended to allow for virtual proximity in lieu of the physical proximity which would typically constitute the paradigm's basis. With "distributed" pair programming, team members continue to work on the same design and coding phases simultaneously, however online collaboration and communication tools are heavily employed. This enables them to share the same editor and environment while sitting at their own computers (though possibly even at different locations), and videoconference to collaborate and exchange ideas in real time.

Research indicates that distributed pair programming (DPP) or virtual pair programming (VPP) produces comparable software to that developed using side-by-side pair programming (Baheti & Williams 2002; Baheti et al. 2002).

While my study is a qualitative study aiming to record the variations in student perceptions of pair programming, most related research found on pair programming evaluates its impact on student performance in programming courses based on final exam scores, on student confidence and enjoyment of the programming process, on personal satisfaction, on the quality of the work produced as well as on the persistence in the computer science major in a quantitative way. (Lai 2011; McDowell et al. 2003; Mendes et al. 2005)

William and Kessler (2001) reported that 95% of the students felt confident when pair programming and 84% enjoyed this experience more.

In a study using a different perspective, (Thomas et al. 2003) attempted to categorize students into three distinct attitudes towards programming: code-warriors, code-a-phoebes and in-betweens. They then teamed up students of differing attitude categories to engage in collaborative (pair) programming. Results showed that students with less self-confidence seemed to enjoy pair programming the most, while stronger programmers seemed to appreciate it the least. Students with the same level of confidence, when paired with students of similar attitude, performed their best.

McDowell (2003) also supported that students working together with a partner will learn more than individual students struggling on their own. (McDowell & Hanks 2003)

As far as time saving and efficiency are concerned, research on pair programming has shown that a shared pool of knowledge between peers helps them handle problems better and faster than students working alone. (Williams et al. 2000)

Negative effects of pair programming have also been reported in related literature. Vanhanen and Lassenious argued that pair programming results in better design, but less functionality and programs of lower quality, and that groups exhibit lower productivity, especially when new teams were formed (Vanhanen & Lassenius 2005). Stephens & Rosenberg (2003), while building their case against extreme programming, observed that a phenomenon encountered often is that one team member does all the work while the other often passively just observes.

Last but not least, Shull et al doubt all the empirical studies that attempt to measure the effectiveness of pair programming by saying that the question "are two heads better than one?" is not precise enough to be measured. Additionally, they argue that the effectiveness of pair programming depends on the expertise of the programmers involved and the complexity of the given programming task. (E. F. Shull et al. 2007)

2 Methodology

This phenomenographic study aims to explore the variations in student experiences when using pair programming to solve a programming problem. In order to expose students to pair programming and create an opportunity for them to describe their experiences, volunteers were required to choose their own partner and work collaboratively on developing a software program, using a plug-in for Eclipse that enables screen sharing. Upon conclusion of their projects, I collected results using semi-structured interviews, which were then analyzed phenomenographically.

Since my inquiry was qualitative in nature and my intent was to explore the different ways in which students experience pair programming and thus gain insight on the phenomenon studied, phenomenography was deemed as being the most appropriate research approach. According to Martin (1995), phenomenography is a research approach that "aims to reveal the qualitatively different ways in which something is experienced" (Martin 1995, p. 166).

Saljo (2007), analyzing Martin's interpretation of phenomenographic research, claims that: *"The prime interest of phenomenographic research… is in finding and de limiting the variation in ways of experiencing reality. It is assumed that there is a limited number of ways of experiencing reality and the description of variations in this respect is the main aim of phenomenography and what makes it a worthwhile exercise. Since ways of experiencing obviously have to be accounted for in language, the phenomenographer describes his object of inquiry by means of what is referred to as categories of description"*

The findings of this study show the range of differences in student perceptions regarding pair programming's effectiveness and applicability.

2.1 Participants

Ten undergraduate college students volunteered to take part in the study, signed a concept form (see Appendix 1) and agreed their participation to remain anonymous. The group consisted of six male and four female students, all of whom had successfully completed two Java programming courses prior to the study. All students were attending a small college in Greece at the time, and all were majoring in Computer Information Systems.

It was made clear to the students that the purpose of this study was not to evaluate their performance and knowledge in Java, but rather to document their feelings and experiences with pair programming, collaborative problem-based learning and their possible effectiveness on accomplishing a common target.

2.2 Research Design

Initially, I contacted twenty of my last semester students, who had successfully completed the second Java programming course (Object Oriented Programming), presented them with the nature and goals of the study at hand and inquired as to their willingness to participate. Ten of them answered positively.

Then, I conducted an informative session on the concepts of pair programming and the software tools and technologies required. The informative session took place at winter session break of 2012, during which students had no classes to attend. Participants were given a brief 15 to 20 minute presentation of pair programming (the presentation was based on Williams and Kessler's paper "All I Really Need to Know About Pair Programming I Learned in Kindergarten") (Williams et al. 2000) and they were instructed on how to use the communications perspective (part of the Eclipse Communication Framework). DocShare (org.eclipse.ecf.docshare) is an Eclipse Communication Framework plugin which implements real-time shared editing. and enables multiple team members to share the editor, share the run-time environment and utilize common debuggers. The source code resides on one team member's computer (in accordance with the typical pair programming metaphor in which the driver "has" the code), while other members can jointly type-in. If members of a team want to work alone at some point, the source code can be uploaded to a repository and shared among team members using a version control system. In the same informative session, participants were then asked to pair up at their own discretion in order to form five final groups. All participant groups received a common programming assignment and were given a one-week deadline to submit completed work for purposes of evaluation and feedback. The assignment consisted of requiring students to implement a missing Java method in an otherwise fully functional card playing game. The purpose of the java method was to evaluate each player's hand and algorithmically determine the winner. Source code for the rest of the program was to be provided and guaranteed as to be working correctly. The level of difficulty of the programming assignment was purposely medium, so that participants would not be frustrated or overwhelmed by the technical challenges of the task at hand and could focus instead on the collaborative aspect and on how they can work together to solve a problem.

Although my research design might appear experimental

2.3 Data Collection

At the end of the study, student opinions were collected via semi-structured interviews. In all, two sets of interviews were performed. In the first set, each student was interviewed separately, while, in the second set, students were interviewed as work groups. Ten individual interviews and 5 group interviews took place. Each student was assigned a participant number, so that anonymity would be kept. At some point in the group interviews, students were also asked to discuss ideas underlying collaborative programming and then discuss how closely their own experiences were aligned with that understanding. All interviews were recorded and subsequently transcribed.

2.4 Validity

As Saijo (1997) stresses in his paper "Talk as Data and Practice: A Critical Look at Phenomenographic Inquiry and the Appeal to Experience", there are many alternative ways of interpreting the data collected from oral interviews given by participants in a study. Thus the categories/conceptions derived from my study can only be considered as subjective and based on my own constructions of meaning, influenced by my personal experiences as a human being. If the same interviews were performed and/or analyzed by another researcher, there is a considerable probability that the derived conceptions could be different.

In order to assure validity of the outcome space, I asked a colleague to verify that my derived categories could be inferred from the transcribed interviews. If there were more time available for this project, I would have asked two or more colleagues to derive their own categories of conceptions from the transcribed interviews and examined how closely their interpretations coincided with my own.

3 Conceptual Framework

My research is based on the social constructivist philosophical view. Social constructivism gives emphasis on the use of peer collaboration and problem-based learning as an instructional method (Prawat & Folden 1994), whereas the constructivist approach advocates that teaching and learning should involve hands-on activities and practical sessions through which knowledge can be built. In the same context, Lave and Wenger (1991) stress the importance of collaboration among learners and the exchange of ideas within and even across communities of practice.

Collaborative learning has been found to have a positive impact on enhancing the ability to work collaboratively with others, on self-esteem and achievement. (D. W. Johnson et al. 2000; Stevens & Slavin 1995). Learning with others is a social process in which students observe their peers as to how they approach problems and find solutions, encourage each other, and, by verbalizing their thoughts, make the process of understanding the situation at hand easier.

Problem-based learning is based on Dewey's philosophical view that practical experience plays a major role in learning (Dewey 1938). Problem-based learning involves contextualizing learning given a "real-world" problem that requires a solution. Students work in small groups to solve a problem provided by their teacher. Problem-based instruction aims to promote students' critical thinking, enhance their problem-solving skills, and prepare them for their future practice or professional endeavors. Programming at the professional level requires individuals to work in teams, collaborate and share knowledge to ensure the success of all those involved. Such real world programming requires extensive communication and collaboration with customers, end users, system analysts, database designers, network architects and many other specialties.

4 Findings

One important assumption of this study was that all participants had the same academic preparation required to accomplish the assigned programming task and that all had similar grades in the completed courses. All student volunteers found the study interesting both in theory and in practice and were more than willing to participate. The unit of analysis was students' conceptions about the effectiveness of pair programming in the areas of enjoyment, quality of work produced and self-confidence.

The result of this study, as with any phenomenographic study, was to form different categories of conceptions found in the meanings of participant interviews and the relationships among them. According to Marton, categories of description form a "way of describing a way of experiencing something" (Marton, 1995. p175)

After analyzing the interviews in this study, 4 qualitatively distinct conceptions of the effectiveness of collaborative programming through pair-programming were identified, reflecting the students' experiences of the pair programming process. The aim was to form a hierarchy of logically related conceptions of the different ways students experienced the task. It should be stressed, however, that all interviews showed more than one of these conceptions – thus, a web of multiple interrelations, rather than a strict hierarchy, was actually observed in this study. I placed the derived conceptions in three categories: Problem Solving, Participation and Coding.

4.1 Category: "Problem Solving"

<u>Conception 1. Collaborative (pair) programming as an efficient way of experimenting with alternative solutions to a problem (to find the best).</u> In conception 1, students focused on how pair programming helped them identify alternative solutions to the programming problem at hand and provided them with effective means of bringing multiple resources ("human minds") to problem solving. Again students were more confident that their chosen solution was the best. The focus of this conception is on quality and development, and the students' perceptions vary from less frustration to confidence.

An interview extract of student conceptions that characterize this category include that of interviewee 8, who focuses on confidence on the solution and speed of development.

Interviewee 8*: …I liked pair-programming, because each time I got stuck, John was there to contribute with his ideas. Brainstorming with your team member during software development can really contribute to finding better solutions, faster* *than working alone…*

Interviewee 3 stresses that although programming is rather frustrating, pair programming reduces the frustration and thus enhances confidence that the solution is correct.

Interviewee 3: …I used to program with my peers in the past, to complete programming assignments because it made the task less frustrating, but I always thought that I was not supposed to… This project helped me realize the reason I liked it in the past: Two brains are better than one!!!!

4.2 Category: "Participation"

<u>Conception 2: Collaborative (pair) programming as enhancing positive competition and motivation between team members in order to produce a solution to a problem at hand.</u>

In conception 2, students argued that pair programming helped them stay dedicated to the task at hand. Proximity (whether physical or virtual), real-time communication, and common focus fostered "friendly" (even unspoken) competition between pair members seeking to actively contribute to their joint success. For example, when one partner would find a solution to a problem, the other was eager (even subconsciously) to do so next. Students also stated that pair programming promoted a team spirit, generating a desire to produce a better product than the other groups. The focus of this conception was participation and collaboration.

The following 2 quotes support the conception of increase motivation to find a solution:

Interviewee 10: *When my partner found a bug, I really wanted to find the next one to prove I was as clever…*

Interviewee 2: *When it comes to debugging, two sets of eyes are better than one. What I could not see, my friend did and vice versa.*

Whereas the next quotes show the variation inside the conception that pair programming increases motivation from completing to task to reaching a personal best.

Interviewer 9: *I wanted our team program to be the best… We did more than the required tasks and we liked it.*

Interviewer 5: *When I was tired, the fact that another person was working with me kept me going…*

<u>Conception 3: Collaborative (pair) programming as being "fun"</u>

In conception 3, students argued that pair programming was more engaging and fun than working alone. To varying extents, the underlying social aspect of pair programming, insofar as it does not constitute a distraction, seem to provide a more relaxing, enjoyable, and ultimately constructive work environment. Student experiences vary from been relaxed, to feeling confident and having fun.

Interviewee 6*: It was the first time I enjoyed programming. I was relaxed (maybe because there were no grades involved) and programming with a friend, chatting and joking when we were stuck helped me code better…*

Interviewee 8*: I found it fun to pair-program especially when my friend and I were working together at night from the confines of our own rooms and w e had Sk ype open so we could communicate as if we were in the same room….*

Interviewee 2: *I love programming because I love solving problems. Solving a pr oblem with a friend lets me feel more confident and share the happiness of accomplishing a task… It's even better when we find together a solution to a problem…*

4.3 Category: "Coding"

<u>Conception 4: Collaborative (pair) programming as a forcing/driving factor to write better-structured and well-documented code.</u>

In conception four, students focus on how the presence of a peer who watched and actively contributed to their code motivated them to adopt better coding styles and stricter coding habits (meaningful variable names, indentation, comments, structure, less "kludges"), so that their peers could understood what they were doing and were able to contribute more effectively. Student experiences from this conception vary from using name standards and using indentation to fully documenting their code. The focus of this conception is structure.

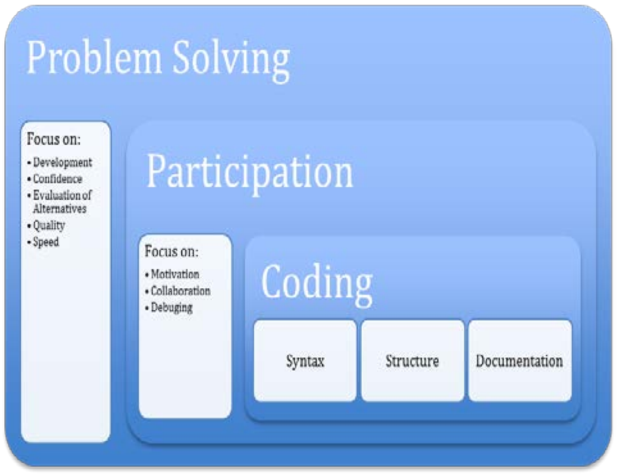
An example from an interview to support this conception is the following:

Interviewee 1: *It was the first time in my programming experience that the variable names I used were not x, y and z. Each time we had to create a variable, we spent time exchanging ideas on how the variable would be named best!*

The above student focuses on the coding style and naming standards, whereas Interviewee 5 stresses that code documentation helped his group collaborate effectively.

Interviewee 5: *We actually documented our code using comments… and not because it was a requirement but to be able to collaborate more efficiently. Sometimes we made fun of it... but, at the end, we were able to remember and understand everything we had written…*

Logical relationships between the 4 conceptions describing student experiences of distributed pair



programming are shown in the outcome space. Each conception subsumes those in a nested circle of the outcome space, which depicts how thinking about pair programming varies qualitatively between and within students. The outcome space is a system of logically interrelated alternative forms of conceptions and their varying internal structures. (Renstrom et al. 1990; Marton & Pong 2005)

Figure 1: the outcome space

The interrelated categories depict the qualitatively different ways by which students understand pair programming. In the inner level "coding" the focus is narrow and detailed whereas in the second category "participation" the focus is on student collaboration to achieve their common goal. Student conceptions vary from making the program work to actually having fun on the process. Most of the students that shared this conception also adopted the coding standards for their development. Finally, the most inclusive category "problem solving" subsumes the presence of participation, and coding all at a level at which students will perform on their best. Not only the will solve the problem but they will also search for the best possible solution.

4 Limitations

The fact that students volunteered to participate in this study, and that they have all historically demonstrated good academic performances, make my sample non-representative of the general population. Additionally, the fact that each member had the option to choose his/her team member made the collaboration easier. The "feel good" factor of pair compatibility as described by (Muller 2004) plays an important role in the performance of the team. When incompatible partners pair program together, they typically dislike the process. (Winkler & Biffl, 2006)

Another factor that conceivably may have had an impact on the students' perceptions of the process was that students were not to be evaluated on the successful completion of the program. This contributed to making the whole programming task less result-oriented and thus less demanding or stressful. Additionally, the amount of work required to complete the assignment was rather limited compared to typical course projects. Finally, students were on winter break, without any course load, so finding common time to collaborate was not as difficult as it would have been in normal semester situations.

Finding common time to collaborate has been reported in related research as a severe drawback of pair programming (Sanders, 2001)

4 Conclusions and future research

As an information technology educator with an emphasis on teaching programming at different levels (introductory, intermediate and advanced), one of the many challenges I face is to make the process of programming as interesting and fun as possible and at the same time prepare my students for the real-world software development environment. With the outburst of Internet technologies, it's imperative that we familiarize our students with different ways of using the Internet other than visiting social networking sites.

This study, within the limits discussed above, showed that is possible to use Distributed Pair Programming to satisfy the requirements of coursework assessment and keep students motivated and satisfied by the process. These results could form the basis for future research; by combining the above-perceived benefits with the time it took students to complete the assignments and the quality of the work produced.

References:

[1] Baheti, P. & Williams, L., Exploring pair programming in distributed object-oriented team projects. Educator's Workshop, 2002.

[2] Baheti, P., Gehringer, E. & Stotts, D., Exploring the efficacy of distributed pair programming. In Extreme Programming and Agile Methods — XP/Agile Universe. Springer Berlin / Heidelberg, 2002, pp. 387-410.

[3] Beck, K., Extreme programming explained: Embrace Change. Reading, MA: Addison, 32(10), 1999, pp.70-77.

[4] Bruffee, K., Collaborative Learning and the 'Conversation of Mankind'; College English, 46(7), 1984, pp.635-652.

[5] Dewey, J., Experience and Education: the 60th Anniversary Edition 1998th ed., West Lafayette, Ind. : Kappa Delta Pi., 1938.
[6] Fowler, E.M., Beck, K. & Cunningham, W., Aim, Fire. Ieee Software, (September/October), 2001, pp.87-89.
[7] Johnson, D.W., Johnson, R.T. & Stanne, M.B., Cooperative learning methods: A meta-analysis. Minneapolis, MN: University of Minnesota, 2000.
[8] Lai, H., An experimental research of the pair programming in java programming course. International Conference on e-Education, Entertainment and e-Management. 2011, pp. 257-260.
[9] Marton, F. & Pong, W.Y., On the unit of description in phenomenography. Higher Education Research & Development, 24(4), 2005, pp.335-348.
[10] Matthews, R., Cooper, J. & Davidson, N., Building bridges between cooperative and collaborative learning. Change: The Magazine of Higher Learning, 27(4), 1995, pp.35-40.
[11] McDowell, C. & Hanks, B., Experimenting with pair programming in the classroom. ACM SIGCSE Bulletin, 2003, pp.60-64.
[12] McDowell, C., Werner, L. & Bullock, H., The Impact of Pair Programming on Student Performance, Perception and Persistence. Proceedings of the 25th International Conference on Software Engineering. Washington, USA: IEEE Computer Society, 2003, pp. 602-607.
[13] Mendes, E., Al-fakhri, L.B. & Luxton-reilly, A, Investigating Pair-Programming in a 2nd-year Software Development and Design Computer Science Course. Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education. 2003. NY, USA: ACM.
[14] Miller, K. & Peterson, R., Creating a positive climate:Cooperative learning. Safe and Responsive Schools. 1980.
[15] Muller, M., An empirical study about the feelgood factor in pair programming. International Symposium on Software Metrics, 1980.
[16] Nosek, J.T., The case for collaborative programming. Communications of the ACM, 41(3), 1998, pp.105–108.
[17] Oxford, R.L. & Dean, E., Cooperative Learning, Collaborative Learning, and Interaction: Three Communicative Strands in the Language Classroom. The Modern Language Journal, 81(4), 1997, pp.443–456.
[18] Panitz, T., Collaborative versus cooperative learning: A comparison of the two concepts which will help us understand the underlying nature of interactive learning. Cooperative Learning and College Teaching, 1997, pp.1-15.
[19] Prawat, R.S. & Folden, R.E.,. Philisophical Perspectives on Constructivist Views of Learning. Educational Phychology, 29(1), 1994 pp.37-48.
[20] Renstrom, L., Andersson, B. & Marton, F., Students' Conceptions of Matter. Journal of Educational Psychology, 82(3),1990, pp.555- 569.
[21] Sanders, D., Student perceptions of the suitability of extreme and pair programming. Proceedings of XP Universe, 2001.
[22] Shull, E.F. et al., Are two head better than one? On the Effectiveness of Pair Programming. IEEE Software, 2007, pp.7-10.
[23] Slaten, K.M. et al.,Undergraduate student perceptions of pair programming and agile software methodologies: Verifying a model of social interaction. Agile Conference, Proceedings, IEEE, 2005, pp. 323–330.
[24] Slavin, R.E. & Madden, N.A., Cooperative learning models for the 3 R's. Educational Leadership, 47(4), 1989, pp.22-28.
[25] Stephens, M. & Rosenberg, D., Extreme programming refactored : the case against XP, Berkeley, California: APress, 2003
[26] Stevens, R.J. & Slavin, R.E., The Cooperative Elementary School : Effects on Students' Achievement, Attitudes, and Social Relations. American Educational Research Journal, 32(2), 1995. pp.321–351.
[27] Thomas, L., Ratcliffe, M. & Robertson, A., Code warriors and code-a-phobes: a study in attitude and pair programming. ACM SIGCSE Bulletin, 35(1), 2003, pp.363–367
[28] Vanhanen, J. & Lassenius, C., Effects of Pair Programming at the Development Team Level : An Experiment. Empirical Software Engineering, 2005, pp.336-345.
[29] Williams, L., But, Isn't That Cheating? 29th ASEE/IEEE Frontiers in Education Conference. 1999, pp. 26-27.
[30] Williams, L. et al., Strengthening the case for pair programming. Software, IEEE, 17(4), 2000, pp.19–25.
[31] Winkler, D. & Biffl, S., An empirical study on design quality improvement from best-practice inspection and pair programming. Product- Focused Software Process Improvement, 2006. pp.319–333