Pair Programming as a Modern Method of Teaching Computer Science

I. Nančovska Šerbec, B. Kaučič, J. Rugelj
University of Ljubljana, Ljubljana, Slovenia

Abstract—**At the Faculty of Education, University of Ljubljana we educate future computer science teachers. Beside didactical, pedagogical, mathematical and other interdisciplinary knowledge, students gain knowledge and skills of programming that are crucial for computer science teachers. For all courses, the main emphasis is the absorption of professional competences, related to the teaching profession and the programming profile. The latter are selected according to the well-known document, the ACM Computing Curricula. The professional knowledge is therefore associated and combined with the teaching knowledge and skills.**

In the paper we present how to achieve competences related to programming by using different didactical models (semiotic ladder, cognitive objectives taxonomy, problem solving) and modern teaching method "pair programming". Pair programming differs from standard methods (individual work, seminars, projects etc.). It belongs to the extreme programming as a discipline of software development and is known to have positive effects on teaching first programming language.

We have experimentally observed pair programming in the introductory programming course. The paper presents and analyzes the results of using this method: the aspects of satisfaction during programming and the level of gained knowledge. The results are in general positive and demonstrate the promising usage of this teaching method.

Index Terms—**teaching method, pair programming**

I. INTRODUCTION

In the paper we analyze the achievement of the selected set of competences which define the knowledge and skills of computer science teachers, especially those, related to teaching programming. Among the competences characteristic for profile of teachers of computer science, there are generic competences [3] and subject-specific competences for the field of computer science [5].

Teachers' competences generally comprehend expert and didactic knowledge and skills from the subject field and pedagogical and psychological knowledge for teaching. They are defined (according to [21]) as a dynamic combination of knowledge, understanding, skills and abilities. Our students achieve competences through courses in the two-subject study program of computer science at the Faculty of Education. Competences are formed in various course units and assessed at different stages. Generic competences are common for all teachers while subject specific competences are characteristic for computer science teachers [5]. Specific competences for the field of computer science are selected in accordance with the document called Computing Curricula [2]. The document's authors are representatives of well-known international computer science associations, such as the IEEE Computer Society and the Association of Computing Machinery (ACM). Computing Curricula document defines »the body of knowledge«, which should be mastered by the graduate students of computer science, describes the core of the graduate computer science programs, defines the teaching goals, suggests models of syllabus and describes the subjects inside them.

Significant topic in the education of future computer science teachers is teaching programming. Programming and teaching programming is realized through different subject all through the studies. After the Bologna processes, the introductory course on programming will be realized in the first year of studies of the computer science at the Faculty of Education, Introduction into programming. Students' knowledge will be deepened through different obligatory courses (such as Programming 1 and 2, Software design, Algorithms and data structures) and optional courses (Java and web programming, Information systems in education etc.).

In the paper we present the didactical models for teaching programming [8]. We explain our experiences with introduction of modern pedagogical methods of teaching, such as pair programming (PP) [9], [11]-[15], [17]-[20]. PP is an agile technology for software development and an important feature of extreme programming [10]. From pedagogical aspect, PP relates to collaborative learning and constructivism.

PP has been subject of several controlled experiments in the last few years. It was reported that it is good for "simple problems in simple code." For simple problems, PP seems to lead to fewer mistakes than solo programming [17], [18]. It was suggested [20] that the code produced by the paired teams was easier to read and to understand. This facilitates detection of errors and maintenance. According to the literature [15], PP is promising form of work, because students working in pairs earn exam and project scores equal to or better than solo students, they have a positive attitude toward collaboration and are significantly more likely to be registered as computer science-related majors one year later. Findings in the literature [14] also suggest that paired students continue to be successful in subsequent programming classes that require solo programming.

In our case, PP experiment was realized in the introductory course of programming. We carry out web poll supported by web-assessment. We evaluate the results of the application of the PP method from different aspects: personal satisfaction of students, their further motivation for programming, self-confidence of programmers, and the assessed level of knowledge that resulted from collaboration in pairs.

II. DIDACTIC MODELS FOR PROGRAMMING

Information and communication technology (ICT) and educational software are integrated into educational processes nowadays. Although ICT is an important topic in teacher education [1], programming has been from its early days considered as a difficult discipline to teach [6], [7], [8], [16], [19]. Kaasbøll [8] reports that, world-wide, first year programming courses in the university suffer a rate of failure between 25 % and 80 %. Observations of Pedroni are similar [16]. Researchers from the fields of didactics and cognitive psychology are exploring the reasons of failure and they found the main reasons laid in the imperfections of the didactics models, which were used for teaching programming in the past. Searching on successful didactic models for teaching programming at the university level was subject of many modern researches [7], [8]; [16]; [14], [19]. The term "didactics of programming" is used in wider frame in this article.

The didactic model is aimed at being [6]:

- a meta-model to be taught to students, so that they can verbalize more of their learning,
- a model for teaching to be taught to the tutors in a course, such that they can align their activities with the lecturers,
- a basis for formulating research questions for further studies of programming teaching.

Teaching programming at the faculties of education is based on comprehension of the programming language syntax, study of semantics, algorithmic thinking, skill of programme writing, learning from examples, "problem solving", programming in the group and teaching programming. In the practice we use the combination of the following models:

Semiotic ladder is based on the language-like features of programming languages. The teaching and learning sequence starts out from syntax, proceeds to semantics and continues to pragmatics of the language-like tools. Its rationale is that syntactical knowledge is needed to express anything, and therefore it should precede the learning of meaning of the language constructs. When the meaning is known, the students can start to learn how to use language for specific purposes, which is the pragmatics. [6], [16]. This model is logical continuation of the earlier way of learning. On the university level, this model is unfortunately not sufficient for adoption of algorithmic thinking which is important competence for programming;

Cognitive objectives taxonomy has used a teaching strategy that resembles Bloom's taxonomy of cognitive objectives. The sequence of instruction comprised using an application program, reading the program, and changing the program. "Creating a program" activity may also be added. This model is suitable for future computer science teachers because it implicitly explains the meaning of algorithms. In [16] the possible use of this model for object-oriented programming teaching it is explained;

Problem solving is strongly motivated by the constructivists' view of how learning occurs. "Through solving problems, the students should extend their experience and repertoire of practice, and the basis for the process is the knowledge structure of the field of programming. The problem solving process is guided by methods and environments. Compared to the previously mentioned approaches, this one stresses the input and outcome of the learning process in terms of knowledge and personal behavior."[16] There has been a four-stage problem-solving process developed that has been used to teach computer science First, try to understand the problem by structuring, dividing, clarifying, and finding sample Input/Output. Second, design a solution by finding related problems and solutions and checking the related solution against the sample Input/Output. Then, write the final solution by completing and adapting the found solutions to the problem. Finally, review the solution by testing it and summarizing what has been learned.

In practice, we use combination of the explained models. In the introductory course are used combinations of first two methods. In the advanced courses, the problem solving model is used.

Further in the paper we will explain our experience with application of pair programming as a modern method of pedagogical work used in an introductory course of programming.

III. PAIR PROGRAMMING AND EXTREME PROGRAMMING

Pair programming (PP) refers to the practice where two programmers work together at one computer, collaborating on the same design, algorithm, code, or test [15].

Computer science faculties are increasingly experimenting, either formally or informally, with PP in the classroom with the intention of its inclusion in the set of techniques that comprise collaboration. At the same time PP is an agile software development, or a special case of extreme programming (XP) [10]. Extreme Programming is a discipline, so called "light methodology", of software development. It is meant for small groups and it works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

PP is used for collaborative programming. The very nature of pair programming implies a psychological and social interaction between the participating programmers and thus brings into play a unique element that we do not see with the conventional individual programming model. According to the literature, a comparison between alike and opposite groups confirmed that the productivity of the opposite group was greater than that of the alike group [12]. Principles of PP coincide with principles of collaborative learning:

- Individuals and communication are important.
- Working software is more important than complete documentation.
- Including (collaboration) of participant is more important then contract negotiation.
- Considering the changes is more important then following the schedule.

In this form of pedagogical work, the participants develop abilities, skills and values, like communicability, simplification, feedback, courage and confidence. These are important for further professional development of the future professors of computer science.

A. *Pair programming state of the art*

The pair is made up of a driver, who actively types at the computer or records a design, and a navigator, who watches the work of the driver and attentively identifies problems and makes suggestions. Both are also continuous brainstorming partners [15]. Rules of behaviour are defined. An effective pair programming relationship is very active. The driver and the navigator communicate, if only through utterances, at least every 45 to 60 seconds. Periodically, it's also very important to switch roles between the driver and the navigator (in our experiment every 30-45 minutes or after the task is finished).

Nosek in his early PP research in 1998 studied professional programmers, 5 individuals and 15 pairs, on a database consistency check. The time needed to complete the task was limited to 45 min. All pairs outperformed the individuals. Although the average time for completion was more than 12 min longer for individuals (41%), the difference was not statistically significant on the 5% level [18].

Williams [13] in her early experiences in 2000 indicated that coupled programmers in the average are 15 % slower then solo programmers but they produce 15 % less errors. In the programming error handling is very important matter and they could be expensive in the phase of debugging and testing.

Nawrocki and Wojciechowski in their research in 2001 [9] studied 21 computer science students on the first four assignments of the Personal Software Process programming course. The students were divided into three groups: one of them used PP. Overall, the PP group was not faster than the other two groups. This result stands in contrast to the Williams et al. and the Nosek studies. But the variability within the pair programming group was smaller than within the other two groups [18].

All I Really Need to Know I Learned in Kindergarten By Robert Fulghum, 1988
Share everything. Play fair. Don't hit people. Put things back where you found them. Clean up your own mess. Don't take things that aren't yours. Say you're sorry when you hurt somebody. Wash your hands before you eat. Flush. Warm cookies and cold milk are good for you. Live a balanced life – learn some and think some and draw and paint and sing and dance and play and work every day some. Take a nap every afternoon. When you go out into the world, watch out for traffic, hold hands and stick together. Be aware of wonder.

Figure 1. Web rules of PP behavior [13]

A larger study in 2007, by Arisholm et al [4], showed: 48% increase in correctness for complex systems, but no significant difference in time, whilst simple systems had 20% decrease in time, but no significant difference in correctness. Overall there was no general reduction in time or increase in correctness, but an overall 84% increase in effort.

A research in 2006 by Liu and Chan [11] presents a rigorous scientific experiment in which novice–novice pairs against novice solos experience significantly greater productivity gains than expert–expert pairs against expert solos.

Williams et al. [15] in 2003 and Bipp et al. in 2008 [20] investigated the advantages of pair programming for educational purposes. They found the quality of the developed code is higher and the integration of less experienced team members is easier.

B. *Inspiration for pair programming*

Our experiences from teaching programming at Faculty of education showed that students implicitly practiced PP without being aware of that. Because the most of them are not experienced in programming and state of the art from PP confirmed the positive experiences for beginners [14], [11], [17] we found that PP could be used in the course of introductory programming. The important aspect of the method is also the collaborative nature of work involved in the PP which is one of the foundations of the education of future computer science teachers, computer science engineers, who work at educational organizations, or software developers, who work in project teams.

C. *Advantages and disadvantages of PP*

As a consequence of PP introduction, according to the literature [14], [11] we expected the following advantages/disadvantages:

Advantages	Disadvantages
- More discipline - Better code (less errors, easier to understand) - Flexible software development - Knowledge interchange between the partners - Pleasant atmosphere - Mutual ownership of the sources - Supervision - Cohesion in the team of two (in the pair participants became more familiar) - Pair is less sensitive on disturbances from environment - We need less computers (PC-s or workstations)	- Giving instructions to the less experienced is tiring - Experienced programmers rather work independently and they fill uncomfortable in the pair - Experienced programmer produces code without (or with less) bugs and it is purposeless to be paired - Is difficult to compare pair with solos empirically - Differences in the programming styles cause conflicts - Par could program less hour/day in comparison with solos which influence the deadline - In the SW enterprises where programmers work at home PP is difficult to realize

We can see that the most of disadvantages are related to engineering environment and consequently we did not expect them to appear in educational environment. So we expected positive experiences from PP application in the introductory course of programming.

1. Time framework of PP?
2. Which positive experience with PP would you put out?
3. Which negative experience with PP would you put out?
4. Please, take few minutes to solve the web adaptive test: http://thirdeye.ath.cx/index.php. Is you result better/equal/ worse then the former, wider test?

Figure 2. Web poll on pair programming experience

D. *Pair programming used in the introductory course of programming*

The PP experiment ran in the Software practicum course at the Faculty of Education, University of Ljubljana (first year of the two-subject study Computer science). In the second part of the course (which is going to be transformed into the Introduction to programming course in our new Bologna curricula) we introduce our students into software development. Students are previously instructed in flow-charts as well as in the syntax and the semantics of programming language Pascal. According to the didactical model of teaching programming, called cognitive objectives taxonomy (mentioned in section 2) students are trained to change programs and write their own code.

Before we start with PP experiment we realized web testing (http://thirdeye.ath.cx/index.php) with purpose to assess students input knowledge. After that students were directed to read the Williams and Kessler well-known paper "All I Ever Need to Know about Pair Programming I Learned in Kindergarten "[13]. We continued with discussion about the rules of behavior during the pair programming (Figure 1). Except in two case students were coupled in 27 pairs by themselves. Pair chose their own names. By random generator we assigned each pair different exercise to be programmed in the following two hours. One week later pairs presented their programs and they .expressed their experiences with PP followed by pro et contra discussion. They were asked to answer the web poll (Figure 2, URL: http://skala.pef.uni-lj.si/irena/ankete/programiranje_v_paru/).

The web was fulfilled by 31 of 54 student involved in the PP experiment.

E. *Analyze of the web poll results*

Web poll was realized to take the students' site about PP and to estimate whether the PP contributed to improve the students' understanding of some programming concepts. Students' knowledge was estimated in two points: before and after the PP method application. From the poll we tried to find out whether the students observed the rules of PP.

As the most positive experience the students set out the broadening of programming aspect of the individual in the pair. As the most positive experience they set out the reconciliation of work and needs of individuals, the communication and, some situations, exhausting and time-consuming programming because of inexperienced individuals.

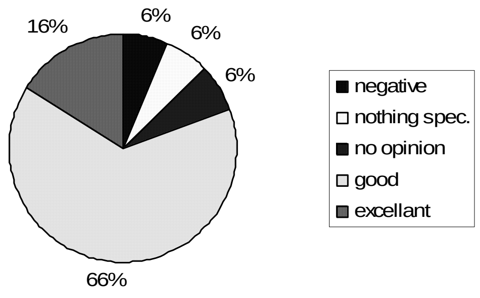

Figure 3. Web poll about PP, general review of the method

For us it is interesting to analyze the general review of the PP method (Figure 3). We found that 82 % had positive experience with PP or they founded PP good or excellent. At the same time 60 % of the students who participated the PP experiment showed better results on the testing after the PP implementation by solving the quick, adaptive test, which could mean that better understanding of programming concepts was achieved.

IV. CONCLUSION

Efficient teaching programming and quality teaching are big challenges. Many teachers doubt in students' competence achievement after the introductory programming courses. In spite of all that, many students in some way fulfill their obligations. This situation could be solved by implementation of new didactical and pedagogical methods.

In this paper we present the most popular didactical models for teaching programming and we explained a collaborative form of work, called pair programming.

PP is an important feature of extreme programming. Our own experiences with PP with groups of students showed that students found useful to work in pairs. In particular, estimating the efforts to realize a feature and distinguishing between essential and nice-to-have features were very difficult for the students. Still, we noticed that the students do benefit from PP.

PP as a collaborative method is good introduction into project work. Students not only express pleasure but they achieved even better results in the control testing. From educational point of view we found PP useful also because it develops competences related to collaboration, integration and knowledge adaptation.

With this in mind, we decided to continue using PP in our courses and to investigate the benefits and drawbacks of this method in more detail.

REFERENCES

[1] A. Lavrič, Osnove visokošolske didaktike (program za izpopolnjevanje): Uporaba informacijske komunikacijske tehnologije v izobraževanju, gradivo za udeležence, 4. del – V modul, CPI, FF, Ljubljana, maj 2007 (in Slovene).

[2] ACM, Computing Curricula 2001, Computer Science — Final Report, The Joint Task Force on Computing Curricula, IEEE Computer Society Association for Computing Machinery, ACM, 15. 12. 2001.

[3] C. Razdevšek Pučko in J. Rugelj, Kompetence v izobraževanju učiteljev, Vzgoja izob., 37(1), 34–41, 2006 (in Slovene).

[4] E. Arisholm, H. Gallis, T. Dyba et al., Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise, Software Engineering, IEEE Transactions on Software Engineering, 33(2), 65–86, 2007.

[5] I. Nančovska Šerbec i J. Rugelj, Analiza pridobljenih in zaželenih kompetenc dvopredmetnega študijskega programa »Računalništvo in ...«, Prispevki k posodobitvi pedagoških študijskih programov, Pedagoška fakulteta, Ljubljana, 78–90, 2006 (in Slovene).

[6] J. J. Kaasbøll, Exploring Didactic Models for Programming, Tapir, 195–203, 1998.

[7] J. J. Kaasbøll, Learning Programming, materials for rsubject Informatics didactics, University of Oslo, 2002.

[8] J. J. Kaasbøll, Teaching Critical Thinking and Problem Defining Skills, Education and Information Technologies, 3(2), 101–117, 1998. doi:10.1023/A:1009682924511

[9] J. Nawrocki and A. Wojciechowski,"Experimental Evaluation of Pair Programming", Proceedings of the 12th European Software Control and Metrics s Conference, London, April, 2001, pp. 269–276.

[10] K. Beck, Extreme Programming Explained: Embrace Change, Reading, MA, Addison-Wesley Professional, 1999.

[11] K. M. Lui in K. C. C. Chan, Pair Programming Productivity: Novice-novice vs. expert-expert, International Journal of Human Computer Studies, 64, 915–925, 2006. doi:10.1016/j.ijhcs.2006.04.010

[12] K. S. Choi, F.P. Deek, I. Im, Exploring the Underlying Aspects of Pair Programming: The Impact of Personality, Information and Software Technology (2007),

[13] L. A. Williams in R. R. Kessler, All I Ever Need to Know about Pair Programming I Learned in Kindergarten, Communications of the ACM, 43(5), 108–114, 2000. doi:10.1145/332833.332848

[14] L. A. Williams, K. Yang, E. Wiebe, M. Ferzli, C. Miller, In Support of Pair Programming in Introductory Computer Science Course, Computer Science Education, 12(3), 197–202, 2002. doi:10.1076/csed.12.3.197.8618

[15] L. A. Williams, C. McDowell, N. Nagappan, J. Fernald, L. Werner, Building Pair Programming Knowledge through a Family of Experiments, Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on Volume , Issue , 30 Sept.-1 Oct. 2003, 143 – 152.

[16] M. Pedroni, Teaching Introductory Programming with the Inverted Curriculum Approach, Diploma Thesis, Department of Computer Science, ETH Zurich, 2003.

[17] M. M. Müller, Do programmer pairs make different mistakes than solo programmers?, In The Journal of Systems and Software 80 (2007) 1460–1471. doi:10.1016/j.jss.2006.10.032

[18] M. M. Müller, Two controlled experiments concerning the comparison of pair programming to peer review, In The Journal of Systems and Software 78 (2005), 166–179. doi:10.1016/j.jss.2004.12.019

[19] N. Guibert, L. Guittet, P. Girard, A Study of Efficiency of Alternative Programming Paradigm to Teach the Basics of Programming, 8th IFIP World Conference on Computers in Education, Cape Town, South Africa, july 2005.

[20] T. Bipp, A. Lepper, D. Schmedding, Pair programming in software development teams – An empirical study of its benefits, In Information and Software Technology 50 (2008), 231–240 doi:10.1016/j.infsof.2007.05.006

[21] Tuning Educational Structures in Europe II, Universities' contribution to Bologna Process, Final Report – Phase 2, urednika: J. Gonzalez, R. Wagenaar, Socrates, 2005.

AUTHORS

I. Nančovska Šerbec was with Faculty of Electrical Engineering, University of Ljubljana. She is now with Faculty of Education, Kardeljeva ploščad 16, SI-1000 Ljubljana, Slovenia, (e-mail: irena.nancovska@pef.uni-lj.si).

B. Kaučič was with Faculty of Education, University of Maribor. He is now with Faculty of Education, Kardeljeva ploščad 16, SI-1000 Ljubljana, Slovenia, (e-mail: branko.kaucic@pef.uni-lj.si).

J. Rugelj is with Faculty of Education, Kardeljeva ploščad 16, SI-1000 Ljubljana, Slovenia, on leave from the Jozef Stefan Institute (e-mail: joze.rugelj@pef.uni-lj.si).

This article was modified from a presentation at the 31st International Convention MIPRO 2008 in Opatija, Croatia, May 2008. Manuscript received 11 August 2008. Published as submitted by the author(s).