Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 231–240
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Pair programming in software development teams – An empirical study of its benefits

Tanja Bipp [a,1], Andreas Lepper [b], Doris Schmedding [b,*]

[a] *Organizational Psychology, University of Dortmund, Dortmund, Germany*
[b] *Department of Computer Science, University of Dortmund, Dortmund, Germany*

Received 27 June 2006; received in revised form 24 May 2007; accepted 25 May 2007
Available online 16 June 2007

Abstract

We present the results of an extensive and substantial case study on pair programming, which was carried out in courses for software development at the University of Dortmund, Germany. Thirteen software development teams with about 100 students took part in the experiments. The groups were divided into two sets with different working conditions. In one set, the group members worked on their projects in pairs. Even though the paired teams could only use half of the workstations the teams of individual workers could use, the paired teams produced nearly as much code as the teams of individual workers at the same time. In addition, the code produced by the paired teams was easier to read and to understand. This facilitates finding errors and maintenance.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Pair programming; Empirical software engineering; Quality of software

1. Introduction

Pair programming [23] is an important feature of extreme programming [2]. Our own experiences [3] with extreme programming with groups of students showed that students less experienced in software development were overwhelmed by planning and organizing an extreme programming project. 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 pair programming. With this in mind, we decided to continue using this concept in our courses and to investigate the benefits and drawbacks of this method in more detail.

In the "Software-Praktikum" course (SoPra) at the University of Dortmund, teams of eight students of computer science carry out software development projects. They improve their programming knowledge and study software engineering by applying software engineering methods in projects close to reality.

In accordance with Nosek [20], we define a *paired team* as a team in which each task of software development is done simultaneously at one workstation by two members of the team. There are two roles within a pair [23]: one person is the "driver" who has control of the pencil/mouse/keyboard and is writing the design or code. The other person, the "observer", continuously and actively examines the work of the driver – watching for errors, thinking of alternatives, looking up resources, and considering strategic implications of the work at hand. The observer identifies tactical and strategic deficiencies in the work. The partners in a pair switch their roles periodically. Additionally, the pairs in the team are put together in new combinations frequently to support the distribution of knowledge in the whole team.

Distribution of knowledge through the whole team can be seen as one advantage of this type of team work. If at least two developers work on every task, a paired team can compensate for the loss of an expert for a specific task

* Corresponding author. Tel.: +49 231 755 2436.
E-mail addresses: Tanja.Bipp@udo.edu (T. Bipp), Andreas.Lepper@udo.edu (A. Lepper), Doris.Schmedding@udo.edu (D. Schmedding).
[1] Tel.: +49 231 755 2841.

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.05.006

much more easily. The solutions of the pairs should be better than the solutions of individuals because the combined creativity and experience of both are greater. Pair programming can be seen as a suitable method to ensure quality of code since the observer does a first code review while the driver writes. This helps to avoid and detect defects early on.

Especially in work groups of students, the potential advantages of working in pairs can be very useful. Within student groups in SoPra, the programming knowledge is very heterogeneous. One potential cause is the students' experience in programming, which in some cases is based solely on homework in programming courses at the university, whereas other students have held jobs as Java programmers or user interface designers. A software development method based on "learning from each other" is supposed to be particularly helpful, especially in situations such as these, where prior knowledge is not homogeneous.

Working in teams of two instead of utilizing the work force of two single programmers, the postulated low efficiency of pair programming is one of the most prevailing counter arguments for this type of work organization. It is therefore not surprising, that this approach is met with particularly little acceptance in teams working under great time pressure.

When two developers work together on one task all the time, it is generally expected that a team consisting of pairs needs twice as much time as a team of developers working on their own. These additional costs must be compared to the positive effects of pair programming on software quality, on the climate in the work teams, and a higher level of knowledge in all team members. Results of current research studying the costs and benefits of pair programming are presented in Section 2.

To study the advantages and disadvantages of software development by teams of pairs, we did some extensive experiments in the SoPra with a total of 95 students in summer 2004 and in spring 2005. In Section 3, these studies are described in detail. Section 4 discusses our experiences on the acceptance of the pair programming concept and the work results of the groups are compared. In the conclusion, the relevance of the results for the field of software engineering and the limitations of our experiment are discussed.

2. Studies with paired teams

A seminal study on pair programming was carried out by Nosek in 1998 [20]. Subjects of his experiment were 15 full-time system programmers from a program trading firm working at system maintenance. They were asked to write a small script. Ten programmers worked in pairs and five of them worked individually and served as a control group. When comparing the five solutions of the experimental group with the five solutions of the control group, Nosek found that the readability of the code of the pairs and the functionality of their solutions was significantly higher. Additionally, the pairs were more confident in their solutions and enjoyed their collaborative problem-solving process more. In fact, all groups outperformed the individuals. Although the average time for completion was 30.20 min taken by the pairs and 42.60 min taken by the individuals, the prediction that pairs need less time to solve a problem was not statistically supported. Prior knowledge of the system programmers and quality of the solution (functionality and readability) were highly correlated for both the experimental and the control group.

Williams et al. [22] carried out an experiment in a senior software engineering course at the University of Utah. The students were divided into two groups with nearly equal levels of prior knowledge: 28 students formed the experimental group of pair programmers, 13 students formed the control group of individual workers. Whereas Nosek's experiment was restricted to comparing coding time and quality of code, Williams and her coauthor observed the whole process of software development, analysis, design, implementation, and test. The students completed four assignments over a period of six weeks. The authors compare development time, efficiency, and quality of the results of both groups. The pairs were able to complete their assignments in 40–50% of time spent on the project by the single developers. The quality of the products was measured by counting passed test cases. The four programs of the pairs passed statistically significant ($p < 0.01$) more of the automated post development test cases. Since the programs of the pairs had the same functionality but less lines of code compared to the programs of the teams of individual developers, the authors conclude that the quality of the code of the pairs is higher. The pairs enjoyed their work more because they are more confident in their results. The authors describe their study as quantitative research, although they mostly only show the mean values of their data and not the complete set of raw data with additional descriptive statistics.

McDowell and his coauthors [16,17] received some interesting quantitative results when studying the effects of pair programming in a programming course for beginners with 600 participants at the University of California. They recognized that weaker students in particular benefit from working in pairs. Less pair programmers quitted the course and pair programmers consequently passed their exams more frequently than students of the control group who had to complete their assignments alone. Also, the long term influence of pair programming was impressive as more of the pair programmers took follow-up computer science courses. In comparison to the other studies, the sample of this study is quite large. But it concentrates on programming and does not look at the entire software development process.

Müller [18] and Padberg [19] also compared the work of individuals with the work of pairs instead of building teams of pairs as proposed in extreme programming. They let 38 students at the University of Karlsruhe write two small programs. On average, the students needed 4 h to finish the assignment. Half of the students worked in pairs, the others worked alone. The collected data showed no correlation between the experience level of a pair and the implementation time, but a significant correlation between performance and feel-good factor [19]. The tasks students had to solve were defined in such a way that automated test cases could be used to measure the functionality of programs. High quality of a program was defined by high rate of passed test cases. The average level of correctness of programs developed by pairs was 29% higher than of programs developed by single programmers [18].

All cited authors, besides Williams et al., restrict their studies to programming as one task during software development. Similar to Williams et al., we want to study the entire software development process. In every experiment considered so far, a task was completed by a pair of two developers or by a single developer who worked alone. The tasks were small and the requirements were precisely defined. In these studies, teams of individuals solving one task together are not compared with teams of pairs switching partners and the roles within the pairs. In contrast, we choose a much more realistic experimental setting. Teams of eight students were asked to solve a complex problem. Half of the teams worked in pairs, the others were teams of individual developers.

3. Method

The effects of pair programming were studied in two experimental settings referred to as Study I and Study II, respectively [15]. The main focus laid on testing the postulated effects of the work organization type on performance (paired teams versus teams of individuals). Study I can be seen as a preliminary study, most of the results presented in Section 4 were yielded in Study II.

3.1. Setting/task

The "Software-Praktikum" course is obligatory for all students in the second year of studying computer science at the University of Dortmund. Aims of the course are the improvement of programming skills, to apply software development methods and tools, and to work in a team.

During six weeks, teams of eight students carry out two projects. Each team of students solves the same tasks. During this time, no other classes are scheduled so that the students can work on their projects all day. In Study I, the students realized a card game. The other task was the management of a cocktail bar. In Study II, the tasks were a quiz game and the simulation of the elevators in a multi-story building, respectively.

We use Java as programming language and UML [5] for analysis and design. The software development teams follow a process model [13] based on Unified Process [14] in their projects.

3.2. Subjects

In total, 95 undergraduate computer science students from the University of Dortmund (2nd/3rd year students) participated in our studies, who all had just successfully finished their intermediate examination in software development. All of them received course credits in return for their participation. The students were randomly sampled to the teams of developers (experimental group as well as control group) to control for potential biases, like differences in prior programming experiences.

The present research paper consists of two different samples. Within Study I (2004), 25 students participated and were randomly assigned to one of four groups with six or seven members with different work conditions. Two teams worked within pairs of two whereas the remaining students worked as teams of individuals. Most of the work took place in a laboratory at the University of Dortmund [4], where the groups could be observed and video-taped while working on the assigned projects. Further research questions lead to a second study, which took place in 2005. Seventy participants were split by chance into 9 groups of nearly equal size. Five of nine groups in Study II were asked to work in groups of two. In support of this concept, special work conditions for these students were created. Paired teams were only assigned half the number of workstations of the number of team members. The other teams were supplied with the amount of workstations according to their team size and received no restrictions regarding their work organization.

3.3. Hypothesis

Based on our experiences within Study I, and taking into account the existent literature on the effects of pair programming, we expected several advantages of pair programming concept. First of all, we were interested how the participants in our studies accepted the concept and were able to increase their work performance. Our hypothesis regarding acceptance, quality of products, and distribution of work load are:

- Less experienced students benefit from pair programming because they are better integrated in the software development teams.
- The knowledge about the project and how to do special subtasks is spread through the whole team.
- Code written by a paired team is better, less complex, and easier to read because it is written to be understood by the whole team not only by its single author.

These questions are interesting and new because the before mentioned studies compare only the performance of a pair of two developers with an individual worker, not development teams with complex tasks.

3.4. Measures

To test our hypothesis regarding the influence of work organization on different work performance related variables, we assessed several covariates and performance related variables within the study settings:

- programming skills (programming knowledge prior to SoPra),
- time spent on the task (duration of work periods),
- work behavior,
- general motivation to do work within SoPra, and
- personality factors.

They were either based on observational, objective data or questionnaires. Subjective constructs were assessed by different items within questionnaires before the start of SoPra or while working on the projects. Additionally, participants were asked to fill out a final questionnaire after the completion of the projects to gain information about the work load distribution within work groups. Answers on sole items were only aggregated to a mean scale value, if they reached an acceptable value of reliability (Cronbach's $\alpha > .70$) [9]).

Participants judged their *prior knowledge* and programming skills on the basis of six five-point Likert scaled items (Example: "*I did a lot of programming during my studies.*"[2]). Answers were combined to a mean value (answer range from 1 = little/5 = much knowledge, Cronbach's $\alpha = 0.82$).

To gain information about the *time spent on the task* by different work teams, participants were asked to regularly mark their hours of work on a questionnaire.

To check if the groups really worked within their assigned work conditions, participants in Study II were asked about their *work behavior* (Example: *"Most of the time we worked in pairs in our group."*). Four different items were generated (ratings from 1 = totally disagree to 5 = totally agree) and combined into a common value (Cronbach's $\alpha = 0.90$). Additionally, conclusions from Study I are based on behavioral observations. In Study II, the database is supplemented by objective data (number of logged in persons). *Distribution of work* was measured by single items at the end of SoPra ("*The work load was not equally distributed in our group,*" "*I barely know some parts of the project.*"). Both items were rated on a five-point Likert scale, ranging from 1 = totally disagree to 5 = totally agree. Finally, three items for measuring ambiguity about the work schedule were included in the final questionnaire after completing both tasks, from an adapted German version [21] of the job ambiguity questionnaire by Breaugh and Colihan [7]. Three items were rated on a seven-point Likert scale, ranging from 1 = totally disagree to 7 = totally agree (Example: "*I knew precisely, in which chronological order I had to accomplish my work,*" Cronbach's $\alpha = 0.73$).

To account for the *motivation* of group members within SoPra, participants were asked eight different questions to state their motivation for engaging in activities at the beginning of the SoPra (Example: *"I am looking forward to the opportunity to apply in SoPra the knowledge that I acquired,"* answering range from 1 to 5, low values = high willingness to get involved within SoPra). Whereas it was not possible to combine the answers within Study I into a reliable scale, Study II yielded an acceptable reliability value (Cronbach's $\alpha = 0.82$). Additionally, participants responded to a German version of the NEO-FFI [6]. This test measures the "Big Five" *personality factors* (neuroticism, extraversion, openness to experience, agreeableness, and conscientiousness) with 12 items for each construct.

For mainly exploratory reasons, we included at the end of the SoPra nine items linked to the acceptance or evaluation of the pair programming concepts within the setup of Study II ("*For which task of software development is working with a partner useful?*" e.g., program design, product test).

3.5. Methods

Indicators of task performance were obtained by measuring the amount and quality of the programming assignments. We measured the range and the complexity of the developed software by means of software metrics. Based on the metric suits of Chidamber and Kemerer [8], many software development tools offer facilities to analyse an object oriented program. The metric tools determine the depth of inheritance, the coupling between the objects and their cohesion. Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Additionally, we asked software development experts to inspect the programs of the students and judge the quality.

To compare the results within the paired teams and teams of individuals regarding questionnaire data, we compared values on the individual level (individual workers vs. pair programmers) or on group level (paired teams vs. teams of individuals). Therefore, we conducted several *t*-tests and ANCOVA's [12] to account for differences in mean values between these groups.

3.6. Manipulation checks

Within the two studies, personality, motivation, time spent on the task, and prior programming knowledge were tested for differential effects between the groups, based on their postulated influence on team performance. Despite the random assignment of students to the different work conditions, three of these covariates showed significant differences in Study I (prior knowledge, personality, and time spent on the task). Therefore, the results of Study I are not

[2] Interested readers may obtain a copy of the complete (German) questionnaire from the authors.

presented in detail within the following sections. Data from this study were mainly used to generate our research hypothesis for testing in Study II. Within this study setup, one group (No. 9) refused to work within the assigned conditions pair programming, so that these seven members are excluded from the dataset for the following analyses. The sample size is therefore reduced to 63.

3.6.1. Prior knowledge

By sampling randomly individual students to different groups of working conditions, we wanted to control for several potential influences on performance. Based upon a wide range of programming experiences (besides university curriculum), prior knowledge of the students was distributed heterogeneously within the groups of both studies. This perception of the SoPra organizer was validated by questionnaire data. The distribution of the questionnaire data regarding prior knowledge of the different development teams in Study II is given in Fig. 1.

The descriptive inspection of self-assessed knowledge scores (see Boxplot in Fig. 1) leads to the assumption that prior knowledge is not equally distributed within the development teams. In some teams, prior knowledge scatters more than in others. However, a statistical test of the average knowledge score in Study II revealed no significant mean difference between the individuals in the experimental and control groups ($M_{\text{pairedTeams}} = 2.58$; $M_{\text{teamsOfIndividuals}} = 2.87$; $t(60) = -1.56$, $p > 0.05$). A distorting influence of different programming experiences on performance between the different development teams can therefore be excluded.

3.6.2. Motivation and personality

Team effectiveness is a broad construct and is therefore influenced by numerous factors. In addition to external

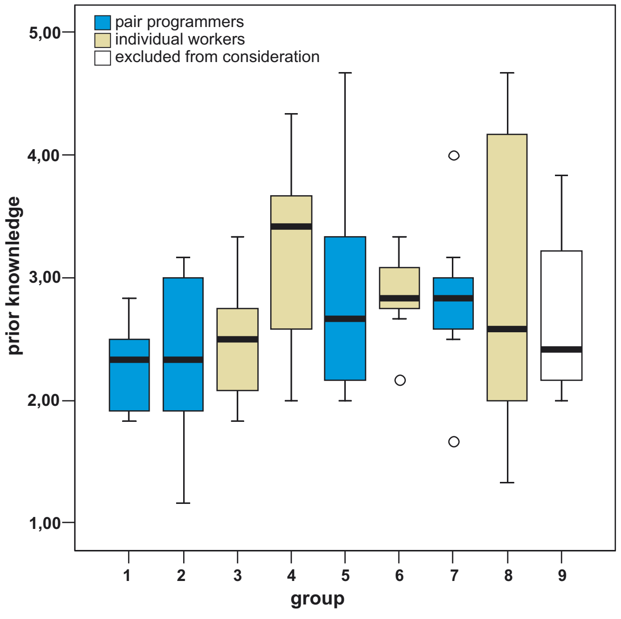

Fig. 1. Prior knowledge within each group in Study II ($N = 72$).

environmental determinants (like work conditions), a lot of research has been done on the composition of teams, taking into account group members abilities, attitudes, and motivation. How team members think about their groups and their goals is likely to have an important influence on team performance. From the observational data and personal discussions with the team members in Study I, we know that they were all highly motivated to complete the assignments. In Study II, the mean values of the ratings of motivation within our questionnaire were close to each other in both experimental groups (1.65–2.25). A statistical test for the difference between these groups revealed no significant effect. The members of both teams were comparably motivated to work on the SoPra assignments.

Whereas the comparison of the Big Five personality factors showed no significant difference between the participants in Study II, members of the paired teams in Study scored significantly higher on conscientiousness compared to members of the groups of individuals. Relating personality to individual and team performance, this factor especially shows intercorrelations with task relevant output variables [1]. This finding raised again considerable doubt about the comparability of the two work setting groups in Study I.

3.6.3. Hours of work

Working hours are usually not specified within SoPra. Instead, every team works the amount of time it needs to fulfill its assignment. Average working time per week varied between 22.7 and 28.8 h between the eight considered groups of Study II. Groups of pairs worked on average 26.0 h a week, whereas the other teams reached a slightly reduced average working time (25.7 h). Therefore, on the mean level the difference between the groups is not statistically significant. On average, all groups worked about the same amount of time per week on their projects. A systematic influence from time spent on the project on performance between the groups can therefore be excluded.

3.6.4. Check experimental setup/work conditions

Within the first study, participants were observed during their work periods on a regular basis. All groups worked in accordance to the experimental work concept, working on their workstations either individually or in twos. In addition to the programming work, all teams held meetings with all members on a regular basis. Answering questions or helping coworkers was not prohibited within the different experimental setups. Such helping behavior occurred under both sets of working conditions. The sample of Study II consisted of far more persons than Study I, so a constant observation could not be realized for reasons of practicability. Instead, the participants were asked about their working behavior in detail after finishing their projects by questionnaire.

Fig. 2 shows the mean levels of the four items used in Study II (higher values indicating an intensified working in pairs). On average, the members of the paired teams

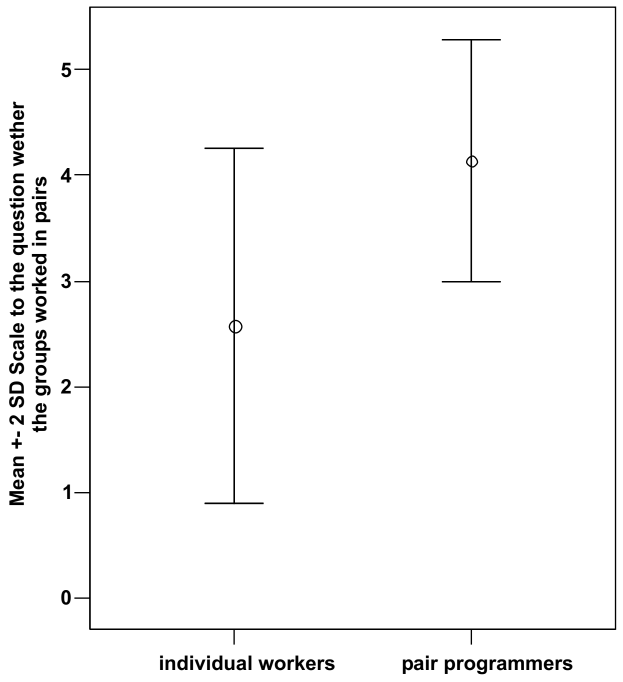

Fig. 2. Fulfilment of paired-team concept in Study II.

reached a value of 4.14, whereas the other groups reached a mean value of 2.58 or the work behavior. The teams of individual workers were not forbidden to work with a partner and did so from time to time. In total, the groups designed to work in pairs realized this working behavior more often than the other groups (significant mean difference), which supports a successful manipulation of work conditions.

Besides the collection of subjective data by questionnaires, the sum of logged in persons within the computer labs was registered by a Unix script in different time periods. Fig. 3 displays the difference between the paired teams

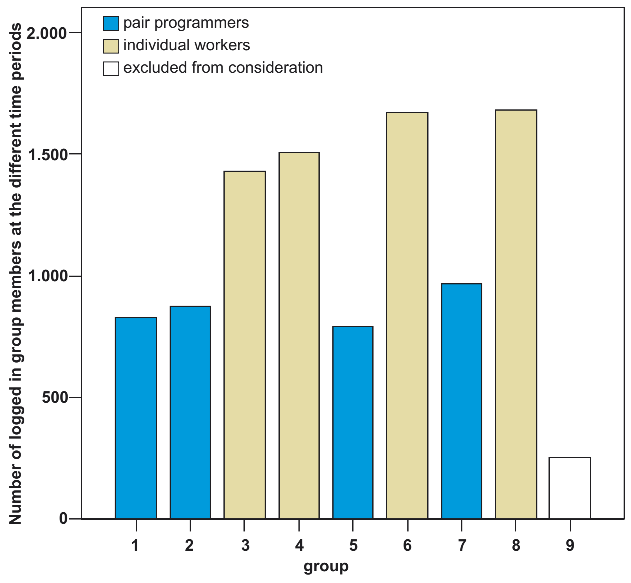

Fig. 3. Number of group members logged in.

(dark shaded) and individual teams (light shaded) clearly. One of the groups (No. 9) refused to participate in the study and worked most of the time on their own laptops without logging in on the provided workstations. Within subsequent analysis, this group is excluded from further consideration.

4. Results

Within the following sections, results regarding the acceptance of the pair programming concept, quality of developed software and further results regarding questionnaire data are presented.

4.1. Acceptance of pair programming

We have to confess that one of our five groups (chosen randomly to work as a paired team) refused to work in this way. Two members in this group (No. 9) expressed great disaffirmation against pair programming already in the first questionnaire before the experiment started. Although the remaining members of this team were absolutely open-minded about pair programming, we could not include this group in our experiment. Pair programming in a team can only be successful if every member of a team accepts this concept.

The remaining participants in the paired teams were asked to report their experiences in the experiment using questionnaires. They judged the role of their partner in the pair in the following way (see Fig. 4): 90% regarded his or her partner as very helpful, 50% felt motivated by their partner, 38% felt hurried by their partner. To feel in a hurry can positively influence performance if it leads to a higher concentration on the task. But this feeling can also influence performance in a negative way, if one feel pressed and controlled. Some participants expressed definitively negative feelings. For example, 5% of the pair-programmers felt hindered or unsettled by the partner.

Nearly all participants in the paired teams agree to the statement "*Working in pairs helps to find errors earlier*." The statement "*I feel uncomfortable while programming because my partner is observing me*" is not well agreed, also the statement that the partner is hindering. Answering the question: "*For which task of software development is working with a partner useful?*," we received the following answers (The students were allowed to give more than one answer.): 83% suggested working with a partner when looking for errors, 79% for program design, 73% for GUI design and two-third for the development of UML diagrams and for complex programming tasks. Most of the nine proposed tasks of the software development process were suggested to be done by pairs. The least accepted was "*every programming task.*" Only 36% would prefer to do every programming task with a partner together. In light of the fact that in complex software development projects there are a lot of simple programming tasks, which

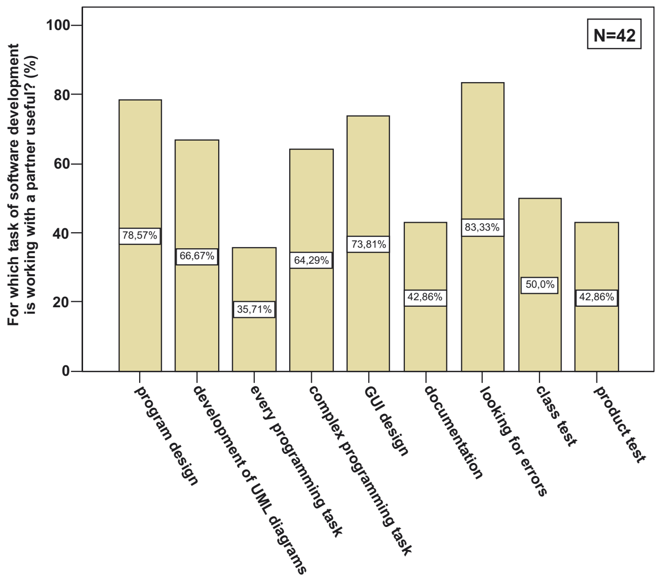

Fig. 4. Answers to the question "For which task is working with a partner useful?".

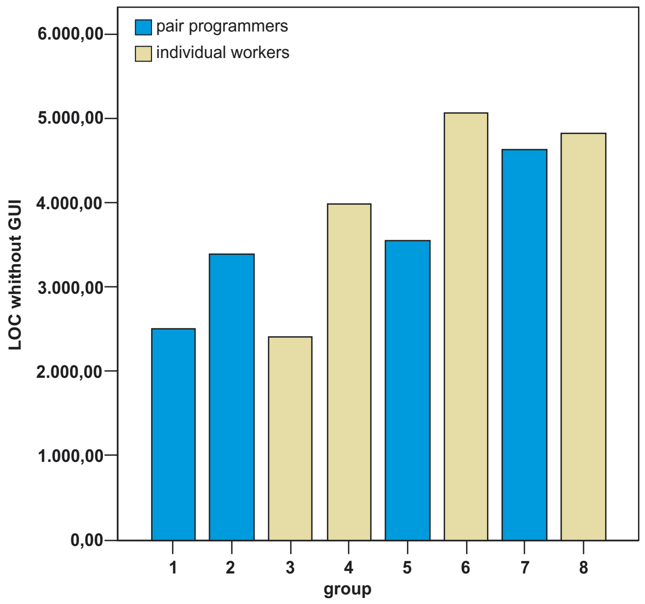

Fig. 5. Range of project for each group.

can easily be done by a single member of the team, this answering quote seems comprehensible.

In conclusion, we can state that the participants who had done their project in a paired team had a lot of positive experiences and suggested it for many tasks of software development. One group did not accept the pair programming concept as two members were against it. This is a risk in any real software development setting.

4.2. Quality of software

Besides the effect of programming in paired teams on quality of the developed software, the influence of the experimental setup on the distribution of work load within the groups was examined.

At the end of the SoPra assignment, a review of the developed programs is done by the organizers of the SoPra together with the participants. All developed programs possessed at least the expected functionality and fulfilled the requirements defined in the description of the problem which should be solved. The produced solutions differed a lot in user interface design and in the offered additional functionality. In both studies, one of the two developed programs was a game. Some of these games simulates team-mates with a strong strategy, one game shows all possible moves to the player, one explains the rules of the game exceptionally colourful, etc. The differences in functionality could not be measured objectively. But the organizers of the lab got the subjective impression that in the second study, the teams of individuals developed programs with a little more functionality compared to the paired teams.

Figs. 5–9 show results of the second study achieved by analyzing the programs of the groups by means of metrics. The metric "lines-of-code" was used to measure the amount of program code. A positive correlation between

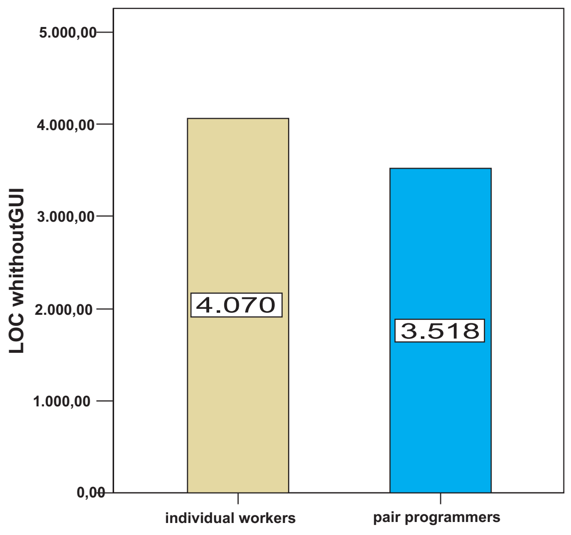

Fig. 6. Average range of projects for paired teams and teams of individual workers, respectively.

LOC and the amount of work hours performed has been proved [10]. The classes of the graphical user interface (which are normally the largest classes in our projects) are developed by means of a graphical user interface editor. Because most lines of code in these classes are generated by this tool, we ignored these classes when counting the lines. Also, we did not include empty lines and lines containing only comments.

One can see in Fig. 5 that the range of the projects scatters between about 2500 lines and about 5000 lines. One paired team and two teams of individual programmers have developed large programs with about 5000 lines of code. Medium scaled projects with about 3500 lines of code were built by two paired teams and one team of individual

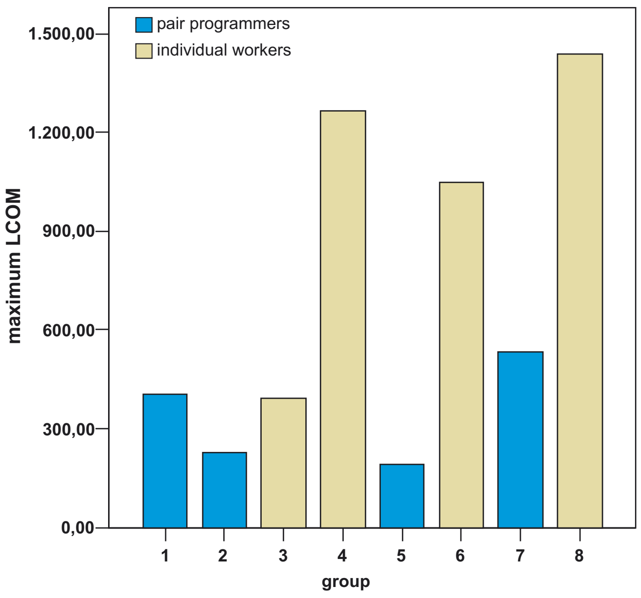

Fig. 7. Program complexity measured by LCOM.

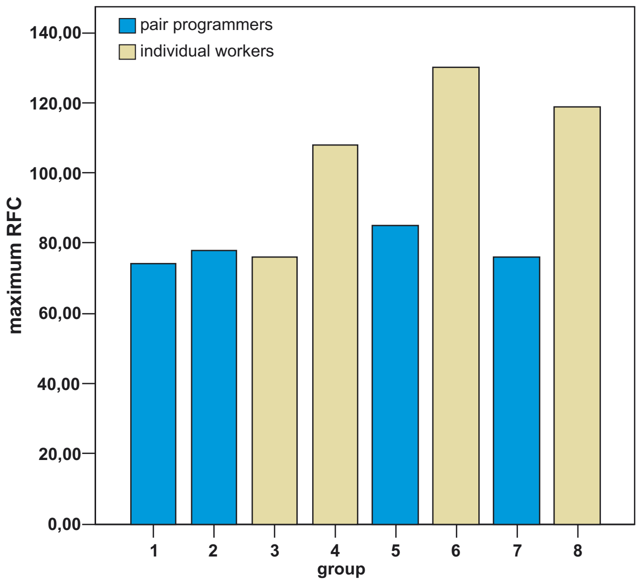

Fig. 8. Program program complexity measured by RFC.

developers. Of each work organization team, one belongs to the cluster of small projects.

In Fig. 6, one can see that the paired teams produced only 13.6% less code at half of the workstations than the teams of individuals in comparable working time. In Study I, the paired teams were able to develop larger programs than the two teams of individuals. But we assume that this was a result of the higher prior knowledge of the students in these groups.

Other research studies (see Section 2) used test cases to estimate the quality of the developed software. This was possible because these projects were small and the given problems were well specified. But in this way, you can only measure the functional correctness of a

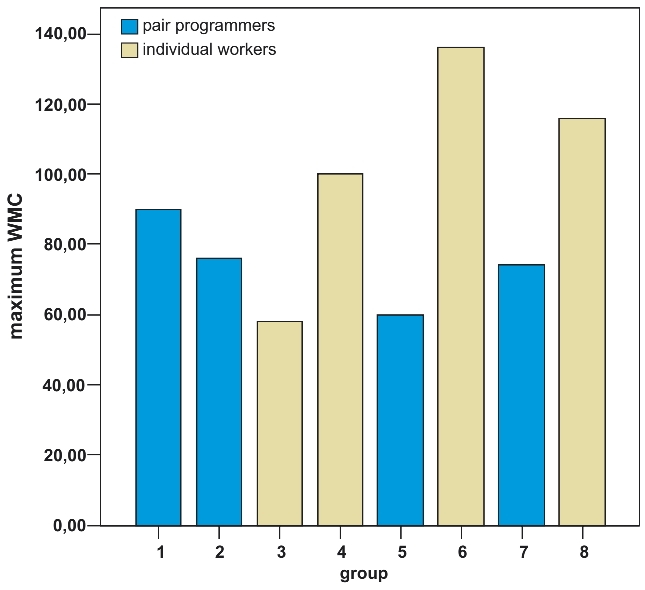

Fig. 9. Program complexity measured by WMC.

program, other aspects of quality, for example, usability of the user interface or the readability of the code, are not considered.

Since we expect that pair programming positively influences the quality of the written code, we were very interested in studying this quality aspect. Especially, we wanted to analyse the complexity of the program code. Very complex code is not easy to understand, therefore maintenance is difficult. Complex code may still contain errors which had not yet been discovered and which may occur later when the program is already in use. Well written object oriented code is simply structured because using the object oriented concepts inheritance and polymorphism leads to clearly structured design [11].

Analysing the produced software by means of metric tools lead to the result that teams of pairs and teams of individuals do not use inheritance differently. The metrics DIT (Depth of Inheritance Tree) and NOC (Number of Children) do not provide different values for both types of teams. The metric CBO (Coupling Between Objects) showed some differences between the teams but we cannot detect a specific trend for pair programmers. In [15] you can find further details.

But we were able to detect some difference studying the complexity of the programs. LCOM (Lack of Cohesion in Methods) (see Fig. 7), RFC (Response for a Class) (see Fig. 8), and WMC (Weighted Methods per Class) (see Fig. 9) differed obviously for both types of teams. A high LCOM value is an indicator for parts of code in a class which do not really belong to this class. A high RFC value serves as indicator to a strong connection of one class to another class. This has to be avoided because it makes changes of the program difficult. If you change the class with the high RFC value, you probably will also have to change other classes. A high WMC value indicates that a class cannot be tested easily because the number of paths of control flow is very large. The three teams of individual workers, team 4, team 6, and team 8, reached high levels of WMC in contrast to the paired teams (see Fig. 9). Only one team of individual workers (team 3) had a similar non complex program. Their program was also the smallest program.

Although the Figs. 8 and 9 show an obvious difference between the paired teams and the teams of individuals, this is statistically not significant because the sample is too small to draw generally applicable conclusions. But we can state that there is a trend to less complex programs for paired teams.

We measured the code quality not only by means of metrics but we also asked computer science experts, colleagues at our software engineering department, to judge the quality of the programs. Based on "*bad code smells*" defined by Fowler [11], a catalogue of criterions was established to guide the software experts. Especially, understandability and readability achieved by using self-explanatory identifiers and well written comments, for example, were considered because the meaning of names and comments cannot be measured by metrics. Each of the software experts judged two classes of some of the teams not knowing if the team was a paired team or not. The code of the paired teams was rated a little bit better. Its readability and understandability were somewhat higher.

Since the members of a paired team do not only program together with their partner but also carry out every other task together during the software development, it would be interesting to compare not only the developed software of both kinds of teams but also the quality of the documents and diagrams produced in the early phases of software development. But since the judgement on the quality of the software proved to be so complex and difficult, we abandoned the idea of considering UML diagrams and documents. The evaluation of diagrams is expected to be more difficult. We are still working on this topic.

4.3. Further results

We recognized in the laboratory setting of Study I that teams of individual developers were likely to exclude less experienced members of their team. Often less able students were not integrated in the team work and it was nearly impossible for them to offer contributions to the project. The fittest students in the teams of individuals always solved the most important tasks. The less experienced students did some less critical jobs, like writing an user manual. From the viewpoint of the team, this organization of work is efficient and useful. But in a learning environment, this kind of team organization should be avoided in order to give every team member the chance to learn new topics.

To set these observations on a solid empirical basis, we included in Study II several items tapping into this aspect of work or learning behavior. By testing for differences between the two work conditions, significant results were found between participants working in paired teams on two items. Judging the statement "*The work load was not equally distributed in our group,*" the students within the groups of two settings scored significantly lower ($M_{\text{pairedTeams}} = 2.68$) than the members of the other groups ($M_{\text{teamOfIndividuals}} = 3.23$; t-test: $t(59) = -2.19$, $p = 0.03$). Participants in paired settings had been more able to achieve an even distribution of work load between their individual group members.

Results for answers on the statement "*I barely know some parts of the project*" are comparable. Here, the teams of individuals showed a significant higher agreement compared to the paired teams members ($M_{\text{pairedTeams}} = 2.26$; $M_{\text{teamOfIndividuals}} = 3.17$; t-test: $t(59) = -3.35$, $p = 0.01$). The distribution of knowledge about different project aspects was improved by working in paired teams. Support for this effect was also found within questionnaire data of Study I. Independent from prior programming knowledge, members of the paired teams reported a lower uncertainty about how to schedule their work within SoPra. Partners within the pair settings were very clear about the necessary steps in the projects or when to apply which work procedure ($M_{\text{pairedTeams}} = 5.64$). In comparison, members of the individual teams reported significant lower values, even independent from their prior experiences in programming ($M_{\text{teamsOfIndividuals}} = 5.15$; ANCOVA: $F(1,22) = 6.19$, $p < 0.05$).

5. Conclusion and limitations

Why do eight developers at four workstations not need twice as much work time as eight developers at eight workstations to solve the same problem? We conclude from our inquiries and our observations that teams working in pairs with changing partners benefit a lot from this type of work organization because they gain more knowledge of all parts of the project. The paired teams use their work time more efficiently because they concentrate much more on their task. If someone needs help, the partner is always nearby to answer questions. During the development of complex software much time is spent in finding errors. Testing and bug fixing in particular is done much easier by two developers than by one.

All pair programmers assess working together with a partner as very positive. Pair programmers take advantage of the higher quality of their code which is less complex, better to read, and easier to understand. This supports finding errors faster.

Most of the pair programmers underline the benefit of pair programming in the questionnaires, while only few of the students who worked in a paired team gave negative comments. One SoPra team was excluded from the experiment because two members in this team refused to work with a partner. If working as a paired team is not accepted by all team members, it cannot be realized.

We cannot yet answer the exciting question of whether doing every task of software development together with a partner leads to more knowledge about software development. The self-evaluation on the newly acquired knowledge by the participants does not show a statistically significant difference between students who worked with a partner and students who worked alone. All participants of both groups declare to have learned much or very much. However, the pair programmers in our study stated that they gained more knowledge on the entire project they have done together with their team mates. Therefore, it can be concluded that pair programmers have learned more about software development.

On the one hand, the loss of efficiency resulting from pair programming is very small and this is the only disadvantage we have seen. On the other hand, the quality of the developed code is higher and the integration of less experienced team members is easier. Therefore, we emphatically recommend working in pairs for teams of students.

The experiments at the university offered the opportunity to compare two different kinds of teamwork for software development under controlled conditions with a number of teams who did the same projects at the same time. Nevertheless, our study setup and conclusions from it have to keep in mind several limitations. Although we conducted our studies within a lab setup, we were not able to control all aspects of work conditions within the six weeks lasting course. Violations to our intended work settings (e.g., like working in teams for teams of paired workers or working by two within teams of individuals) can therefore be critical for making causal interpretations of our results. We did not see clear signs of diffusion or imitation of the paired team concept within the control groups, but all students saw each other on a regular basis during the study, so that an exchange of ideas and experiences probably took place. To gain clearer results, future studies could, for example, realize a diversified experimental setup (e.g., with waiting control groups) to account for these kinds of problems. Furthermore, we analysed data mainly on the basis of individual statements on the group level. To gain clearer results, future research should concentrate on the possibility doing hierarchical analyses for testing for effects, not only taking into account results at the group but also individual level.

A next step should be to test whether our results can be transferred to industrial software development settings. Limitations to expand our results can obviously be seen within our experimental lab settings, the use of a student sample, a small number of groups and the used tasks which all limit the external validity of our results. Although the integration of the study within a mandatory course within the study program offered us the opportunity to test our hypotheses on a heterogeneous sample (not only volunteers), the experiences show that some people refused to cooperate within the lab setup or the pair programming concept. To specify the causes for this behavior and to replicate our results with different and larger samples and varying operationalization methods of core constructs should be the next steps within our line of research to generate reliable results.

References

[1] M.R. Barrick, G.L. Stewart, M.J. Neubert, M.K. Mount, Relating member ability and personality to work-team processes and team effectiveness, Journal of Applied Psychology 83 (3) (1998) 377–391.
[2] K. Beck, Extreme Programming Explained: Embrace Chance, Addison Wesley, 2000.
[3] I. Beckmann, D. Schmedding, Experimente mit XP in der Lehre, GI Jahrestagung 2 (2004) 122–126 (in German).
[4] T. Bipp, J.Hüvelmeyer, Das INWIDA Labor, Available from: <http://www.inwida.uni-dortmund.de> (in German).
[5] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language – User Guide, Addison Wesley, Reading, MA, 1999.
[6] P. Borkenau, F. Ostendorf, NEO-Fünf-Faktoren Inventar, Handanweisung, Hogrefe Verlag für Psychologie, 1993 (in German).
[7] J.A. Breaugh, J.P. Colihan, Measuring facets of job ambiguity: construct validity evidence, Journal of Applied Psychology 79 (1994) 191–202.
[8] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[9] L.J. Cronbach, Coefficient alpha and the internal structure of tests, Psychometrika 16 (1951) 297–334.
[10] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, International Thomson Computer Press, 1996.
[11] M. Fowler, Refactoring – Improving the Design of Existing Code, Addison Wesley, 2000.
[12] W. Hays, Statistics, Harcourt Brace, Orlando, FL, 1993.
[13] C. Kopka, D. Schmedding, J. Schröder, Der Unified Process im Grundstudium – Didaktische Konzeption, von Lernmodulen und Erfahrungen, DeLFI 2004, pp. 127–138 (in German).
[14] P. Kruchten, The Rational Unified Process: An Introduction, Addison Wesley, 1999.
[15] A. Lepper, Eine empirische Studie über Paararbeit in der Softwaretechnik, Master thesis at the department of computer science at the University of Dortmund, 2005 (in German).
[16] C. McDowell, L. Werner, H. Bulock, J. Fernald, The effects of Pair-programming on performance in an introductory programming course.ACM SIGCSE Bulletin, in: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, vol. 34, No. 1 (2002).
[17] C. McDowell, L. Werner, H. Bulock, J. Fernald, The impact of pair programming on student performance, perception and persistence, in: Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, 2003, 602–607.
[18] M.M. Müller, Two controlled experiments concerning the comparison of pair programming to peer review, The Journal of Systems and Software 78 (2005) 166–179.
[19] M.M. Müller, F. Padberg, An empirical study about the feelgood factor in pair programming, In International Symposium on Software Metrics, Chicago, September 2004.
[20] J.T. Nosek, The case for collaborative programming, Communications of the ACM 41 (3) (1998).
[21] K.-H. Schmidt, S. Hollmann, Eine deutschsprachige Skala zur Messung verschiedener Ambiguitätsfacetten bei der Arbeit, Diagnostica, 44, 1998, pp. 21–29 (in German).
[22] L. Williams, R.R. Kessler, W. Cunningham, R. Jeffries, Strengthening the Case for Pair-Programming, IEEE Software, July/August 2000.
[23] L. Williams, R.R. Kessler, Pair Programming Illuminated, Addison Wesley, 2003.