Proposing Mobile Pair Programming[*]

F Steimann, J Gößner, U Thaden
Institut für Informationssysteme, Universität Hannover , Germany
{steimann, goessner, thaden}@kbs.uni-hannover.de

1 Background

Learning in small groups is deemed highly effective. Given this observation, pair programming appears to be a natural setting for learning how to program. However, even in groups learning takes considerable time and pair programming has some prerequisites that are not easily met, not even in a university setting: it requires suitable facilities where students can work on a computer sitting side-by-side, discussing aloud without disturbing others, and it requires sufficiently long time slots during which students actually have the opportunity to meet. Restrictions of this kind are not only present in a university setting; they equally apply to many industrial scenarios, in which the search for sources of increased productivity and quality dominates over learning issues.

Given these problems and given that more and more (German) universities consider transitioning to a notebook university[1], mobile pair programming based on specially equipped laptops appears to be a viable alternative. Local and motion independence increases the possibility of collaboration, the more so if campuses are widely distributed, students commute long distances or are otherwise separated; at the same time, ad-hoc networks allow collaboration over short distances (eg, for two collaborators travelling in a train).

2 Prior experience with stationary pair programming

2.1 Setup

We have conducted a first small experiment among 60 students of a regular 200 h object-oriented programming project held during the 4th semester of an applied informatics curriculum. Students were asked to divide up into ten equal-size groups, and each group was subsequently provided with four exercises, each comprised of six more or less independent tasks. Students were encouraged to process their tasks as pair programmers (two tasks for one pair) but could also solve them alone (one task per individual). After completion of each task, each student was asked to fill in a questionnaire asking for her/his experience, duration of task processing, and whether (s)he preferred pair programming over working alone. In order to avoid bias based on personal sympathy/antipathy, we required pairs to rotate within each group after each block.

Although we never intended to declare our experiment a serious study, we had at least hoped to arrive at comparative results of some credibility. Unfortunately, outcome was basically incomparable, so that our experiment collapsed to a rather general pair programming acceptance test. However, we'd like to stress here that this was not an inherent flaw in our initial study design, but rather owing to a common software engineering problem: although we had originally intended to provide a complete suit of unit tests with each task and automate the extraction of time and quality metrics, we had to give up on this venture due to personnel shortcomings (beware Boehm's software risk #1).

2.2 Results

In a first quick survey half way through our project, almost three quarters of our students said that they preferred pair programming over working alone. Only 15% explicitly stated that they disliked pair programming. 85% said that they had actually learned something during their pair programming activities; this may be considered rather strong evidence that pair programming is conceived as effective by learners, although proof of actual learning progress is of course lacking, as is that of increased quality or productivity.

[1] not to mention companies equipping their buildings with WLAN and their staff with laptops

[*] presented at: *OOPSLA 2002* Workshop on Distributed Pair Programming

However, as it turned out a detailed analysis of the questionnaires conducted at the end of the project showed that actual pair programming times were much shorter than suggested by the initial enthusiasm: only one third of all tasks were actually solved in pairs. Even though students are obviously open to pair programming, its actual acceptance is disappointingly low; as one student put it, "pair programming only makes sense […] with better opportunities to meet and work together at the face". It seems that meeting is a severe obstacle to (co-located) pair programming.

3 A controlled experiment demonstrating the effects of mobile pair programming

It has been noted that software engineering often lacks the scientific rigor expected from an engineering discipline [4]. This is often excused by the supposed softness of software, its related processes and required skills. Indeed, many scientific endeavours in our field are concluded with a proof-of-concept demonstration, showing that what has been elaborated so carefully in the theoretical part of one's work at least doesn't fail completely in practice. By introducing mobile pair programming, we hope to improve the software process, an achievement that cannot be proven on formal grounds. Therefore, the first step in our investigation must be to come up with a carefully designed experiment.

3.1 Context

We have set up a pilot project (named UBICAMPUS) at our home university in the course of which 260 students will be equipped with (partially sponsored) notebooks within the next year.[2] These students will be obligated to take part in a number of innovative classes building heavily on the concepts of ubiquitous and wireless computing. One of these classes is the software project that has already led to the above described experiences with pair programming. As part of the UBICAMPUS project, we hope to demonstrate improvements in the collaborative (= pair programming) efforts among students and thus in a learning effect which can be attributed to the introduction of wirelessly linked notebooks to the pair programming process. Validity of results will not be bound to a university setting, however; lessons learnt should be applicable to industrial scenarios, too.

3.2 Goal

The goal of our work is to establish criteria and conditions for and to conduct a *repeatable experiment* designed to demonstrate the effect of *independent mobility* (a more general form of dispersal) on the acceptance of pair programming.

3.3 Study design

As indicated earlier, the primary goal of our study is to show that pair programming is facilitated by the possibility of mobile and – if desired – also distant pairing, ie, that students equipped with interconnected mobile laptops "meet" (virtually or real) and collaborate more often and/or for longer periods than those tied to a fixed workplace. Only secondarily we intend to investigate productivity as measured by duration and quality[3] of task accomplishment. More precisely, we want to investigate

> whether or not the acceptance *A* of pair programming in our given software project depends on the mode of collaboration *M*, which may be mobile (*MPP*) or stationary pair programming (*SPP*).

The independent variables of our experiment are:

- difficulty/complexity of each given task
- prior programming skills of the participants
- prior experience of the participants with pair programming

[2] Project sponsor: BMBF Förderprogramm Neue Medien in der Bildung, Ausschreibung "Notebook-University", Förderkennzeichen 08NM222A

[3] In an experiment testing the test-first approach, Müller and Hagner could not find an improvement in software quality, but in software understanding as measured by the reuse of (number of calls to) methods [2].

The dependent variables of our experiment are:

- time needed to solve the given tasks by each individual/pair[4]
- time of actual cooperation among each pair
- quality of the solutions as measured by
  - the number of errors remaining
  - readability and other external metrics (measures yet to be identified)

To determine the effect, we intend to conduct a controlled trial with a control group pair-programming the usual way. Although measurement of interaction among collaborators is no true problem if communication is exclusively carried out over the network, we face severe practical limitations when recording the interaction for the co-located participants, since talking, changing keyboard etc. escapes all practical measurement. We will offer the students a chess clock to record swapping of roles, but there can be no guarantee that this will be used as demanded.

Admittedly, studies of this kind are difficult to conduct, because human factors are invariably hard to account for. However, others have already collected sufficient experience with similar trials [1, 2]; learning from their experience will help us conduct a study with repeatable results.

3.4 Test group

Participants of the test group are equipped with notebook computers with Internet access enabled via WLAN (802.11b), Ethernet cable and modem. MS Messenger is used as the basis for application sharing and voice communication. WLAN access points are distributed over parts of the campus and computer science buildings. Cooperation in peer-to-peer mode is also possible and the connection of choice if participants reside in close proximity, even side-by-side.[5]

Pair programming activities are initiated and monitored through a special tool wrapping the required IDE sharing and communication functionality. Task switches between driver and navigator are recorded as well as individual communication times. The numbers and times of submission of an intended solution to the test suite are also recorded; this is the same as in the control group.

3.5 Control group

Participants of the control group are assigned workplaces and work times in a university computer room, but are also allowed to work at home. Particularly this last point is debatable, since one might argue that in extreme cases – if participants work solely at home and alone – there is no difference in the test and the control group. However, nothing prevents mobile pair programmers from sitting side-by-side in a university room, so that a certain overlap in the conditions is unavoidable. Note that control group participants must co-locate in order to count as pair programmers; thus, our experiment also compares co-located versus dispersed pair programming, only that the latter has a mobile freedom added to it.

4 Evaluation

The experiment will be carried out during the summer semester 2003. The expected number of participants in the hypothesis and control group is 40 students each. Results will be published by fall 2003.

5 References

[1] P Baheti, L Williams, E Gehringer, D Stotts, J McC Smith "Distributed pair programming: empirical studies and supporting environments" TR02-010, Dept. CS, University of North Carolina.

[2] M Müller, O Hagner "Experiment about test-first programming" in: *EASE – Conference on Empirical Assessment in Software Engineering* (2002).

[3] M Müller, W Tichy "Case study: Extreme Programming in a university environment" in: *International Conference on Software Engineering ICSE* (Toronto, 2001) 537–544.

[4] WF Tichy "Should computer scientists experiment more?" *IEEE Computer* 31:5 (1998) 32–40.

[4] Since we want to measure acceptance, we do not require students to work in pairs, but only suggest it.

[5] This could be used for the two-screen pair-programming mode described in [3].