On the Impact of Warmup Phases on the Economics of Pair Programming

Frank Padberg and Matthias Müller
Fakultät für Informatik, Universität Karlsruhe, Germany
padberg|muellerm@ira.uka.de

Abstract

Pair programmers need a" warmup phase" before the pair can work at full speed. We study how large the impact of the lower productivity during warmup is on the business value of a pair programming project. To this end, we extend our net present value model for pair programming to explicitly include a learning interval for pairs. We then carry out a simulation study where we vary the shape of the learning curve, the length of the learning interval, the final productivity level of the pairs, the market pressure, and the size of the workforce. Our simulations show that the cost of the warmup phase is small compared to the project value. This result suggests that the learning overhead is not an obstacle to introducing and using pair programming.

1. Current Interests

Both of us (Frank and Matthias) are interested in the economic analysis of development techniques and paradigms, such as XP. The goal of our studies is to develop guidelines how to use certain technology in software engineering to the best advantage.

Frank also is working on cost estimation and optimal scheduling for software projects. This is a hard problem since feedback between activities introduces a considerable amount of uncertainty into the software process, see [6-9].

Our group in Karlsruhe is known for its long-standing substantial research in empirical software engineering. Matthias has contributed a number of empirical studies about XP to our group's repository, for example, a study which compares pair programming against conventional development with reviews [13,14].

2. Past Work

Frank is an early member of the EDSER community. He contributed papers on probabilistic cost modeling and the impact of product features (such as the strength of the coupling in the software) on the project cost to the workshops [1-3].

Later on, Matthias joined in to work on the economics of pair programming and extreme programming [4,5]. These papers use concepts from finance such as net present value and return on investment. Expanded versions appeared at international conferences [10,11].

3. Issue Statement

Pair Programming (PP for short) is a technique where all tasks are performed by pairs of developers using one keyboard, display, and mouse. The idea is that working in pairs increases productivity and improves the software quality as compared to conventional development. These potential advantages are reached at the expense of a higher personnel cost, though. Hence, the decision to apply PP in a project must be supported by a classical cost-benefit analysis.

There is empirical evidence that developers who program in pairs need a "*warmup phase*" before PP becomes fully effective, that is, before the pair works at full speed [17,19,20]. We have made similar observations in our XP lab courses for computer science graduate students [15,16]. Human factors are of key importance in this context; some developers easily communicate and share ideas, others don't.

Clearly, the lower productivity of a pair during the warmup phase affects the economic assessment of the project. This holds both for developers who have no experience with PP and those who do, but are member of a newly formed pair. We speak of the "*learning phase*" in the first case and the "*startup phase*" in the second case. We expect that the learning phase for PP newcomers will take longer and exhibit a different *learning curve* than the startup phase for experienced pair-programmers.

The questions which we want to study, then, are:

- How large is the impact of the lower productivity during warmup on the business value of a pair programming project?
- How can the cost of inducting personnel into pair programming be minimized?

Type of issue
The questions under study are strategic with respect to the cost of introducing PP in a company for the first time, but operational with respect to estimating the value of individual PP projects and estimating the cost of forming new pairs. The questions relate to business, management, and process issues. The decisions to be made will be mostly non-technical in nature.

Context
The context of the issues studied in this paper are Agile Methods, in particular, Extreme Programming (XP). Pair Programming is a key technique of XP which can be applied independently of other XP techniques (such as test-driven development). Both XP and PP are being recommended especially for projects which are carried out under strong market pressure. Since programmer pairs should finish tasks faster than single developers, projects which use PP should get to market earlier than conventional projects. The resulting gain in market share can – under suitable conditions, see our previous studies [10,11] – more than balance the increased personnel cost.

Stakeholders
Main stakeholders are project managers who….

- must decide from an economics perspective whether to switch to PP in their projects;
- must estimate the business value of their next PP project;
- must induct new personnel into PP with the lowest possible overhead.

We aim at providing project managers with tools for computing the business value of PP projects, methods of tradeoff analysis for PP, and guidelines for using PP. Other stakeholders are….

- developers who must pair-off with new colleagues from time to time;
- researchers who want to understand the tradeoffs involved in PP and who aim at increasing the productivity of pairs through suitable tools and techniques.

Information needs
We need both qualitative and quantitative data about the shape of the learning curves involved in pair programming. In addition, we need numbers for the productivity of pairs for different (industrial) project settings. When applying the model, a manager also needs some estimate of the discount rate appropriate to model the market pressure for his next project.

4. Proposed Approach

Research methods
We have already constructed and used an economic model for Pair Programming in previous papers [10,11]. The model is based on the concept of *net present value* (NPV) and has a number of parameters, including the productivity of pairs, the size of the workforce, the market pressure as modelled by the discount rate, and the size of the software product; see below for details.

Currently, our model assumes that the productivity of pairs is constant throughout the project. We propose to extend this model to explicitly include a learning interval, respectively, startup phase. The productivity of the pairs should increase according to some learning curve until their final productivity level is reached.

Based on such an extended model, we propose to carry out a comprehensive simulation study to analyze what happens to a PP project's buiness value when we systematically vary ….

- the shape of the learning curve and the length of the learning interval;
- the final productivity level of the pairs;
- the market pressure;
- the size of the workforce;
- the size of the product.

Assumptions
The productivity of a pair ususally is expressed as a (fractional) multiple of the productivity of a single developer. The corresponding factor is called the *pair speed advantage* (PSA). The PSA typically ranges between 1.0 (no difference in productivity) and 2.0 (double speed). In this study, we'll make the following assumptions with respect to the PSA:

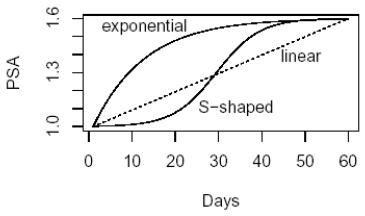
- We assume that a pair gets more and more productive each working day; that is, we assume that a pair's PSA value steadily increases from 1.0 to the final value over time.
- We assume that the growth of productivity during the warmup phase a priori is described by some learning curve. An s-shaped curve assumes that the pair productivity will increase only slowly in the beginning, but later on will increase quickly as the developers get to know each other better. The formula is

  $$\frac{1}{1+b\cdot e^{-ax}} \quad (a>0,\ b>1).$$

  An exponential curve assumes that the pair productivity will quickly increase right from the beginning. The formula is

$$1 - e^{-ax} \qquad (a > 0).$$

Both curves reach a saturation level after some time and are commonly used to describe human learning processes. The next figure shows two examples for learning curves where the PSA grows from 1.0 to 1.6:



- We assume that pairs work with a constant speed after the learning or startup phase; that is, a pair's productivity (and hence, PSA) will be constant after warmup. This is a natural assumption for coarse-grained models such as our NPV-based model.

Process or solution

We quickly summarize our existing economic model for pair programming and its extension with an explicit learning or startup phase.

The net pesent value (NPV) of a project is defined as

$$\mathrm{NPV} = \frac{\mathrm{AssetValue}}{(1+\mathrm{DiscountRa\ te})^{\mathrm{DevTime}}} - \mathrm{DevCost}.$$

With net present value, the money paid by the customer for the final product (AssetValue) is discounted back at a certain DiscountRate. It is common in economics to model strong market pressure by large values for the discount rate.

The development time DevTime (in working days) for a project of size ProductSize (in lines of code) solves the equation

$$\mathrm{DevTime} \cdot \mathrm{Productivity} \cdot \mathrm{PSA} \cdot \mathrm{NumOfPairs} = \mathrm{ProductSize}.$$

The development cost (DevCost) for the project is basically proportional to the development time, number of pairs, and developer salary, see [11].

We now extend the model by assuming that the pair speed advantage varies from day to day. The amount of work accomplished on day t then equals

$$\mathrm{Productivity} \cdot \mathrm{PSA}(t) \cdot \mathrm{NumOfPairs}$$

where $\mathrm{PSA}(t)$ is the value of the pair speed advantage on day t. Therefore, the development time in days for a pair programming project becomes the smallest number DevTime which solves the inequality

$$\sum_{t=1}^{\mathrm{DevTime}} \mathrm{Productivity} \cdot \mathrm{PSA}(t) \cdot \mathrm{NumOfPairs} \geq \mathrm{ProductSize}.$$

When using this inequality, we silently simplify the math a little bit by approximating the learning curve by a piecewiese constant curve (constant for each day).

Finally, the newly computed development time must be converted to (fractions of) years, as this is the unit needed in the NPV formula.

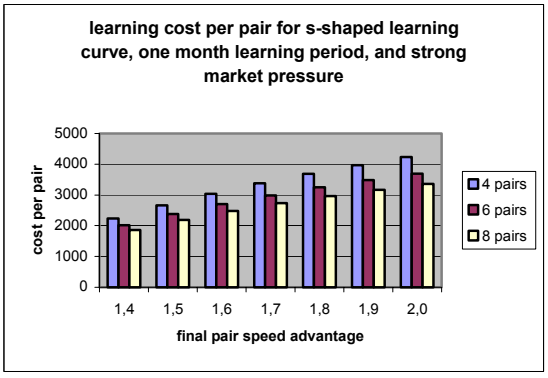
5. Results, Status, Prospects, and Needs

First computational results

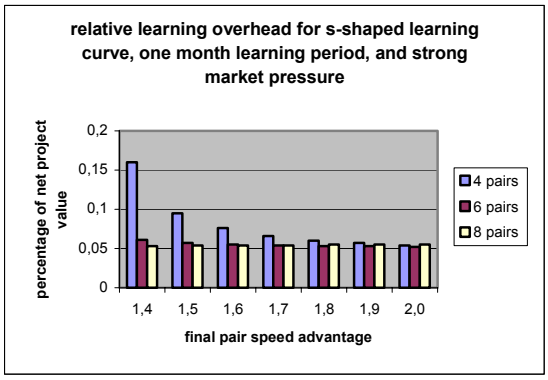
We have already done various simulations of our extended model to study the impact of the lower pair productivity during learning on the PP project value. We present just a few representative results here.

The sample project under study has the following parameters: ProductSize = 25,200 LOC, Productivity = 350 LOC per month, AssetValue = 1,000,000 Euros, DiscountRate = 75 percent (strong market pressure). The workforce level (NumOfPairs) and the final PSA vary.

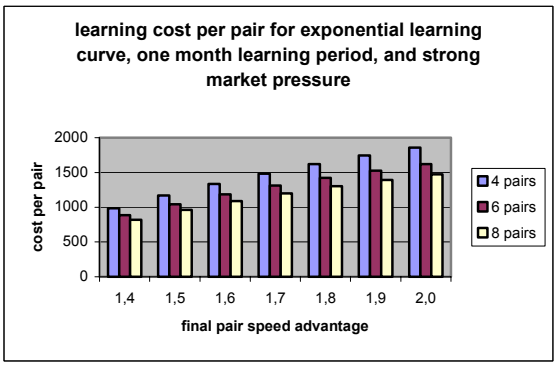
The next figures show the learning cost *per pair* and the relative learning overhead for s-shaped, respectively, exponential learning, each with a learning period of one month.



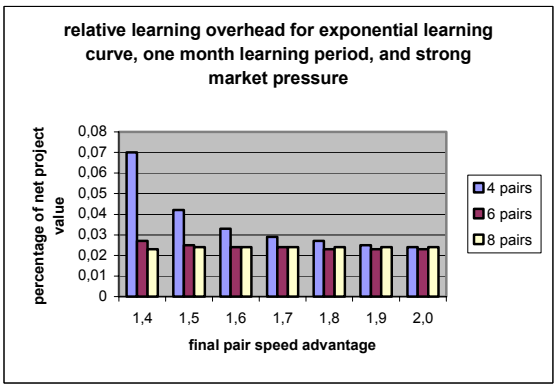
The larger the final PSA, the higher the learning cost per pair. On the other hand, the cost per pair *decreases* as the number of pairs increases. Except for small values of the final PSA in conjunction with a small workforce, the learning overhead is limited:



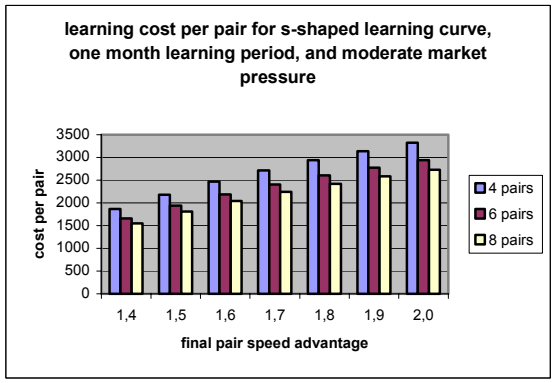
The picture looks similar for exponential learning curves:



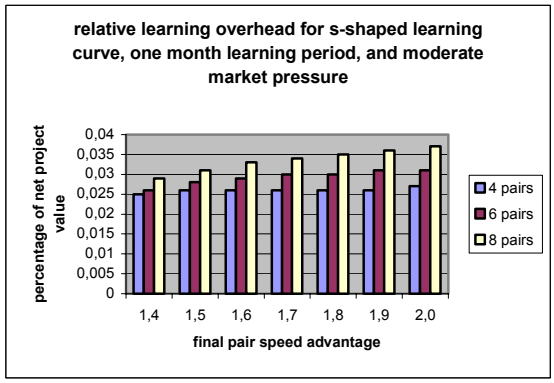
The main difference is that the cost per pair is *only half* as large for exponential learning than for s-shaped learning — note the different scaling of the y-axis. The relative learning overhead behaves similar to s-shaped learning, but again, the percentages are cut in half:



We have performed analogous computations assuming a *moderate* market pressure of 25 percent, instead of 75 percent. The charts for the learning cost per pair look similar to those in the strong market pressure case, but the costs are lower, both for exponential and s-shaped learning. We just give the chart for s-shaped curves:



The relative learning overhead (in percentages of the project value) also is significantly smaller than for strong market pressure. As opposed to the strong market pressure case, the overhead increases with the number of pairs and the PSA value. Again, we just give the chart for s-shaped curves:



Preliminary conclusions

Our simulation results suggest that, from a project-economics perspective, the risk of using PP lies *not* in the overhead associated with the learning period, respectively, startup phase:

- For exponential learning curves and reasonable length of the learning interval, the overhead caused by learning amounts to only a few percentage points of the project value. Recall that we expect exponential curves to be typical for the startup phase of pairs formed of developers who have experience with PP.
- The learning period for developers who have no experience with PP is a one-time cost (except when staff turnover is heavy and new staff consists mainly of PP newcomers). Even when assuming that s-shaped learning curves apply in this case, the overhead is likely to be limited by 5 to 10 percent for the very first PP project, given that management can schedule developers to

learn how to pair-program in a fairly well-staffed project under only moderate market pressure.

As a consequence, we argue that the risk of using PP comes either from developers who for some reason or another are opposed to the idea of working in pairs, or, on the side of management, from overestimation of the speed advantage which can be achieved by using PP in the context of a particular company or project.

As a note to researchers in this field, we'd like to point out that one must be careful when measuring the PSA in experiments. If the project or assigment is too small and developers have had no prior experience with PP, then the value measured will probably be wrong, as it includes the learning phase with its lower productivity. Besides differences in individual skills, this effect might explain why seemingly contradictory values for the PSA have been reported in the literature, ranging from 1.0 to 1.7 [12,17,18,20].

6. Open Issues

Currently, we have no empirical knowledge about the "true" shape of the learning curves for programmer pairs: are they exponential, s-shaped, or other? The same statement holds for the length of the learning interval.

We also need more empirical data about the initial and final PSA value. There is empirical evidence that the PSA value for newcomers ranges between 1.0 [17] and 1.2 [12,20]. There is no empirical evidence for values below 1.0. In this study, we have assumed that the PSA starts at 1.0, but our model also works for initial values different from 1.0.

To collect the empirical data and validate our model, it is necessary to conduct controlled experiments and field experiments with professional developers. The subjects should include both PP newcomers and staff experienced with PP. In the experiments, the productivity of the pairs must be frequently sampled to identify the shape of the learning curve.

7. References

[1] F. Padberg, "A fresh look at cost estimation, process models and risk analysis", *EDSER-1*, May 1999

[2] F. Padberg, "Linking software design with business requirements – quantitatively", *EDSER-2*, May 2000

[3] F. Padberg, "Tracking the impact of design changes during software development", *EDSER-3*, May 2001

[4] M. Müller, F. Padberg, "Extreme programming from an engineering economics viewpoint", *EDSER-4*, May 2002

[5] M. Müller, F. Padberg, "About the return on investment of test-driven development", *EDSER-5*, May 2003

[6] F. Padberg, "A Discrete Simulation Model for Assessing Software Project Scheduling Policies", *Journal SPIP 7:3,4 (2002)*, pp. 127-139

[7] F. Padberg, "Using Process Simulation to Compare Scheduling Strategies for Software Projects", *Proceedings APSEC 2002*, pp. 581-590

[8] F. Padberg, "Scheduling Software Projects to Minimize the Development Time and Cost with a Given Staff", *Proceedings APSEC 2001*, pp. 187-194

[9] F. Padberg, "A Probabilistic Model for Software Projects", *Proceedings ESEC 1999*, pp. 109-126

[10] M. Müller, F. Padberg, "On the Economic Evaluation of XP Projects", *Proceedings ESEC 2003*, pp. 168-177

[11] F. Padberg, M. Müller, "Analyzing the Cost and Benefit of Pair Programming", *Proceedings METRICS 2003*, pp. 166-177

[12] A. Cockburn, L. Williams, "The Costs and Benefits of Pair Programming", *Proceedings XP 2000*

[13] M. Müller, O. Hagner, "Experiment about test-first programming", *IEE Proceedings Software 149:5 (2002)* pp. 131-136

[14] M. Müller, "Are reviews an alternative to pair programming?", *Proceedings EASE 2003*, pp. 3-14

[15] M. Müller, J. Link, R. Sand, G. Malpohl, "Extreme Programming in Curriculum: Experiences from Academia and Industry", *Proceedings XP 2004 (to appear)*

[16] M. Müller, W. Tichy. "Case Study: Extreme Programming in a University Environment", *Proceedings ICSE (2001)*, pp. 537-544

[17] J. Nawrocki, A. Wojciechowski, "Experimental Evaluation of Pair Programming", *Proceedings ESCOM 2001*

[18] J. Nosek, "The Case for Collaborative Programming", *Communications ACM 41:3 (1998)*, pp. 105-108

[19] L. Williams, C. McDowell, N. Nagappan, J. Fernald, L. Werner, "Building Pair Programming Knowledge through a Family of Experiments", *Proceedings ISESE 2003*

[20] L. Williams, R. Kessler, W. Cunningham, R. Jeffries, "Strengthening the Case for Pair-Programming", *IEEE Software July/August 2000*, pp. 19-25

8. Biography

Matthias Müller reveived the diploma and PhD degrees in informatics from the University of Karlsruhe, Germany, in 1996 and 2000. In his dissertation, he worked on the topic of optimizing compilers for parallel architectures. In the last four years, he has focused on software process improvement, especially lightweight software processes.

Frank Padberg received the master's degree in mathematics from the University of Erlangen, Germany, and the Ph.D. degree in computer science from the University of Saarbrücken, Germany. He has been supported by fellowships and research grants from the Deutsche Forschungsgemeinschaft DFG. Before joining the university, he gave seminars within the industry on networking technology and operating systems.