# Extreme Programming from an Engineering Economics Viewpoint

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

**Abstract**

We use the concept of net present value to study for which project settings a lightweight programming paradigm such as Extreme Programming can be an advantage over a conventional development process. If the project is small to medium scale, the product must be of high quality, *and* time to market is the decisive factor, Extreme Programming can be the right paradigm to choose.

## 1 Introduction

Lightweight development processes such as Extreme Programming (XP) are causing a lot of hype currently. Proponents of XP claim that using their development paradigm brings strong advantages over conventional processes, including lower management overhead, higher team productivity, improved software quality, and shorter release cycles. Yet, there is only limited empirical evidence in support of these claims. In addition, as far as we know there currently is no systematic study if and how the potential benefits of XP balance the cost of XP. Such a study must be based on key concepts from *economics.*

In this paper, we study XP from an economics point of view. We analyze the cost and benefit of XP for a sample software project under different settings, systematically varying key project parameters which are influenced when using XP techniques. In the XP community, the techniques of XP are considered to be most beneficial if the requirements are unstable and time to market is a decisive factor. Therefore, we use the *discount rate* in the formula for a project's *net present value* to explicitly take into account the impact of time to market.

## 2 Extreme Programming

Extreme Programming breaks with a number of traditional software engineering practices [1, 2]. The source code is the main documentation; the rational is that writing documentation slows down the developers and is neglected anyway. There is no specification; executable test cases which are written before the code is developed serve as a substitute (*test-first*). There is no separate design or testing phase; design, implementation, and testing go together in small increments. Designing for change is explicitly prohibited; only the simplest possible design satisfying the most important set of features is chosen. However, XP prescribes a combination of special practices instead. All programming tasks must be performed by *pairs of programmers* using one display, keyboard, and mouse. The test cases must be re-run continuously during development. The code must be occasionally re-structured while retaining its functionality (*refactoring*). For more details on XP practices and typical difficulties when implementing XP see [7].

From the *project economics* point of view, the most important practices of XP are *pair programming* and *test-first.* Since developers team up to work on the same task, personnel cost basically is doubled. The main claim of XP is that the increased cost is more than outweighted by three factors:

- A pair of programmers has a higher development speed than a single programmer.
- Continuously checking the code against the test cases improves the quality of the code.
- The code produced by a pair of programmers has a reduced defect rate as compared to a single programmer.

Two studies provide empirical evidence for the advantages of pair programming [3, 8]. A possible explanation for a reduction in the defect rate is that pair programming leads to an ongoing review of the program code by the second developer, and that the two developers can readily share their ideas. The studies also provide some quantitative figures. The $\mathsf{PairSpeedAdvantage}$ of a pair of programmers over a single programmer is claimed to be 29 percent on average [8]. The $\mathsf{PairDefectAdvantage}$ of a pair of programmers over a single programmer is claimed to be 15 percent on average [3].

## 3 Economic Modelling

To compare XP with conventional development, we use the concept of *net present value.* The net present value of a project [4, 5] is defined as

$$\mathsf{NPV} = \frac{\mathsf{AssetValue}}{(1 + \mathsf{DiscountRate})^{\mathsf{DevTime}}} - \mathsf{DevCost}.$$

Since in our study time to market is assumed to be the decisive factor, a delay in the project's completion time will lead to a loss of market share, thus drastically decreasing the value of the whole project. We account for time to market by choosing successively higher values for the $\mathsf{DiscountRate}$ in the formula for the net present value. The $\mathsf{AssetValue}$ is assumed fixed. The XP development practices have a mixed impact on the net present value of a project, as we shall see shortly.

### 3.1 Development time

Figures for the productivity of a developer range between 300 and 350 lines of code per month, including design and coding, but excluding testing [9]. For conventional projects, the development time is

$$\mathsf{DevTime} = \frac{\mathsf{ProductSize}}{350 \times \mathsf{NumberOfDevelopers}} + \mathsf{QATime}.$$

The time needed for quality assurance is special here: it's the time needed to *equalize* the defect rate advantage which XP has over the conventional process, see the next subsection. For an XP project, no additional time for quality assurance is required, but pair programming must be taken into account:

$$\mathsf{DevTime} = \frac{\mathsf{ProductSize}}{350 \times (100\,\% + \mathsf{PairSpeedAdvantage}) \times \mathsf{NumberOfPairs}}$$

$$\mathsf{NumberOfPairs} = \frac{\mathsf{NumberOfDevelopers}}{2}.$$

For both conventional and XP projects we implicitly make the simplifying assumption that the productivity of the developers, respectively, programmer pairs, adds up; we do not take into account any increase in the team communication overhead as the team size increases.

### 3.2 Quality assurance

A programmer typically inserts 100 defects per thousand lines of code [6]. A good conventional software process eliminates up to 70 percent of these defects [6]. Therefore, using conventional development there are

$$\mathsf{DefectsLeft} = \mathsf{ProductSize} \times \frac{100}{1000} \times 30\,\%$$

defects left in the software after coding. In contrast, XP claims to lead to an improved defect rate. In that case, the conventional project must make up for the quality difference of

$$\mathsf{DefectDifference} = \mathsf{DefectsLeft} \times \mathsf{PairDefectAdvantage}$$

defects in a separate quality assurance phase before entering the market. With XP, no separate quality assurance phase is required. The length of the quality assurance phase for the conventional process depends on the time needed to remove a single defect:

$$\mathsf{QATime} \;=\; \mathsf{DefectDifference} \times \frac{\mathsf{DefectRemovalTime}}{135\,\mathsf{h}} \times \frac{1}{\mathsf{NumberOfDevelopers}}.$$

Here, we assume that a developer works 135 hours per month.

### 3.3 Development cost

For simplicity, the development cost is assumed to consist just of the salaries for the developers and the project leader:

$$\mathsf{DevCost} \;=\; \frac{\mathsf{DevTime} \times (\,\mathsf{NumberOfDevelopers} \times \mathsf{DeveloperSalary} \;+\; \mathsf{ProjectLeaderSalary}\,)}{12}.$$

The formula is the same for conventional and XP projects.

## 4 Results

### 4.1 Sample project

To study the cost and benefit of XP, we compute the net present value of a hypothetical sample project under different settings. The fixed parameters of the sample project are as follows:

- The ProductSize is 16, 800 lines of code.
- The DeveloperSalary is 50, 000 dollars per year.
- The ProjectLeaderSalary is 60, 000 dollars per year.
- The AssetValue of the software is 1, 000, 000 dollars.

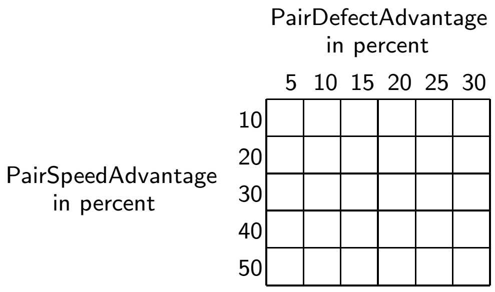
There also are some project parameters which we vary systematically in the study:

| parameter | | | | | | |
|---|---|---|---|---|---|---|
| PairSpeedAdvantage | 10 % | 20 % | 30 % | 40 % | 50 % | |
| PairDefectAdvantage | 5 % | 10 % | 15 % | 20 % | 25 % | 30 % |
| DefectRemovalTime | 5 h | 10 h | 15 h | | | |
| DiscountRate | 0 | 25 % | 50 % | 75 % | 100 % | |

We visualize the difference between the net present value of the conventional project and the net present value of the XP project using different scales of grey:

| | |
|---|---|
| ⬜ | disadvantage of XP |
| ░ | up to 5 percent advantage of XP |
| ▒ | between 5 and 10 percent advantage |
| ▓ | more than 10 percent advantage |

For fixed DefectRemovalTime and DiscountRate, the difference between XP and the conventional project is visualized as a "checkboard" as the parameters PairSpeedAdvantage and PairDefectAdvantage vary:



### 4.2 Four pairs

Assume that the number of developers available for the project is bounded by eight. In that case, XP can run the project with four pairs only. The difference between the net present value of the conventional project and the net present value of the XP project is visualized in Figure 1.

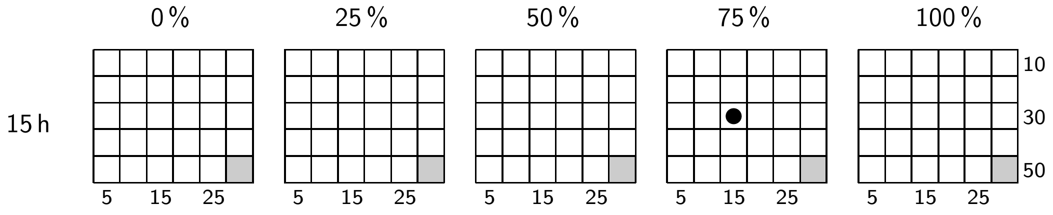


Figure 1: Four programmer pairs vs. eight single developers.

Pair programming will lead to such an increase in development time that – even for high discount rates and high defect removal times – the net present value of the conventional project is larger than the net present value of the XP project. The reason is that only four tasks can be worked on at the same time in the XP project, as compared to eight tasks in the conventional project. This disadvantage is not balanced by the speed advantage that a programmer pair has over a single programmer, nor by the defect rate advantage of programmer pairs. For example, even when assuming that the discount rate is 75 percent per year, the time needed to remove a defect is 15 hours, and the optimistic figures given in the literature for the pair speed advantage (30 percent) and the pair defect advantage (15 percent) actually hold, the value of the XP project is 34 percent smaller than the value of the conventional project, see the field with the solid dot in Figure 1.

### 4.3 Eight pairs

On the other hand, assume that there is a pool of developers available who could be added to the project, but that it does not make sense to subdivide the tasks in the project any further due to the size and structure of the project. In that case, the project leader might choose to run the project using XP with *eight* pairs instead of only four pairs, doubling the personnel as compared to the conventional project. The difference between the net present value of the conventional project with eight developers and the net present value of the XP project with sixteen developers (eight pairs) is visualized in Figure 2.

Despite the increased personnel cost, for high discount rates and high defect removal times the net present value of the XP project is larger than the net present value of the conventional project. The reason is that the speed advantage and the defect rate advantage of programmer pairs come into full play. As a result, the increase in personnel cost is outweighted by the gain in market share. For example, when assuming that the discount rate is 75 percent per year, the time needed to remove a defect is 15 hours, the pair speed advantage is 30 percent, and the pair defect advantage is 15 percent, the value of the XP project is about 6 percent higher than the value of the conventional project, see the field with the solid dot in Figure 2.

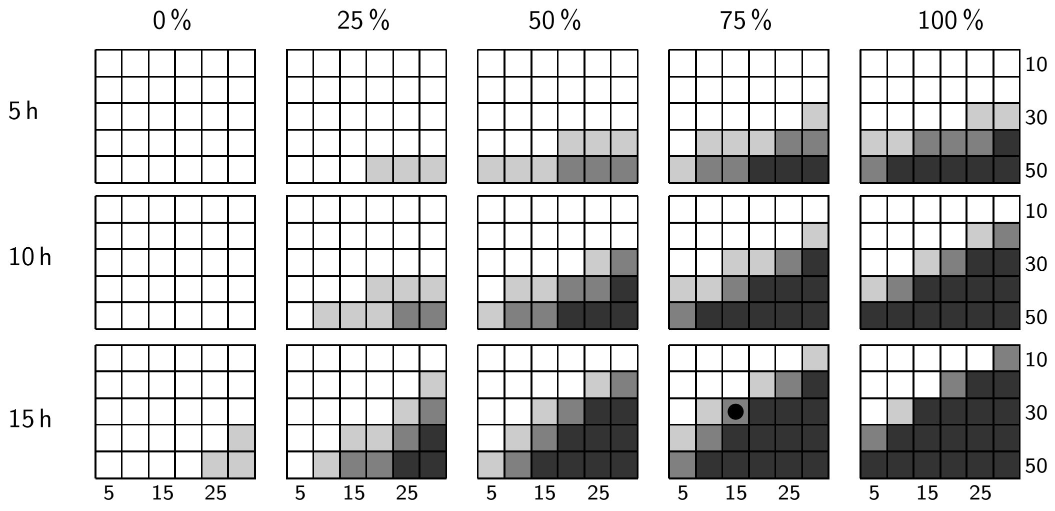


Figure 2: Eight programmer pairs vs. eight single developers.

## 5 Conclusion

The conclusions to be drawn from the results of the computations are clear. As long as there is a good way to assign tasks to additional developers, using XP does not provide an advantage. On the other hand, if the size and structure of a project are such that the tasks can not be reasonably subdivided any further, adding developers in such a way that they team up to pairs can reduce the development time and thus increase the value of the project despite the increased cost for personnel. This holds especially if time to market is the decisive factor. The smaller the market pressure, the more the potential advantages of XP diminish.

## References

1. K. Beck. Embracing change with extreme programming. *IEEE Computer*, pages 70–77, October 1999.
2. K. Beck. *Extreme Programming Explained.* Addison Wesley, 1999.
3. A. Cockburn and L. Williams. The costs and benefits of pair programming. In *eXtreme Programming and Flexible Processes in Software Engineering XP2000*, Cagliari, Italy, June 2000.
4. H. Erdogmus. Comparative evaluation of software development strategies based on net present value. In *First International Workshop on Economics-Driven Software Engineering Research EDSER*, Los Angeles, USA, May 1999.
5. W. Harrison, D. Raffo, and J. Settle. Measuring the value from improved predictions of software process improvement outcomes using risk-based discount rates. In *First International Workshop on Economics-Driven Software Engineering Research EDSER*, Los Angeles, USA, May 1999.
6. W. Humphrey. *A discipline for software engineering.* Addison-Wesley, 1997.
7. M. Müller and W. F. Tichy. Case study: Extreme programming in a university environment. In *23rd International Conference on Software Engineering ICSE*, pages 537–544, Toronto, Canada, May 2001.
8. J. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105–108, March 1998.
9. I. Sommerville. *Software Engineering.* Addison-Wesley, 1996.