Analyzing the Cost and Benefit of Pair Programming

Frank Padberg
Fakultät für Informatik
Universität Karlsruhe, Germany
padberg@ira.uka.de

Matthias M. Müller
Fakultät für Informatik
Universität Karlsruhe, Germany
muellerm@ira.uka.de

Abstract. *We use a combination of metrics to understand, model, and evaluate the impact of Pair Programming on software development. Pair Programming is a core technique in the hot process paradigm of Extreme Programming. At the expense of increased personnel cost, Pair Programming aims at increasing both the team productivity and the code quality as compared to conventional development. In order to evaluate Pair Programming, we use metrics from three different categories: process metrics such as the pair speed advantage of Pair Programming; product metrics such as the module breakdown structure of the software; and project context metrics such as the market pressure. The pair speed advantage is a metric tailored to Pair Programming and measures how much faster a pair of programmers completes programming tasks as compared to a single developer. We integrate the various metrics using an economic model for the business value of a development project. The model is based on the standard concept of net present value. If the market pressure is strong, the faster time to market of Pair Programming can balance the increased personnel cost. For a realistic sample project, we analyze the complex interplay between the various metrics integrated in our model. We study for which combinations of the market pressure and pair speed advantage the value of the Pair Programming project exceeds the value of the corresponding conventional project. When time to market is the decisive factor and programmer pairs are much faster than single developers, Pair Programming can increase the value of a project, but there also are realistic scenarios where the opposite is true. Such results clearly show that we must consider metrics from different categories in combination to assess the cost-benefit relation of Pair Programming.*

Keywords. Pair Programming, Net Present Value, Cost-Benefit Analysis, Extreme Programming.

1. Introduction

Pair Programming is a core technique in Extreme Programming [1, 2]. With Pair Programming, all tasks must be performed by *pairs* of programmers using only *one* display, keyboard, and mouse. Proponents of Pair Programming claim that their development paradigm brings strong advantages over conventional processes, including higher team productivity and improved software quality.

Pair Programming is a technique which aims at making software development more efficient and life easier for the practicing software engineer. In such a case, software engineering research must provide the metrics and models which are necessary to analyze both the benefit *and* cost of applying the technique. For industrial projects, the decision whether to use the technique for the next project can then be supported by a cost-benefit analysis which builds upon the results of research.

In view of the hype about Extreme Programming and other lightweight process paradigms, Pair Programming clearly requires a cost-benefit analysis which is based on objective measures from *economics.* Software can be and has been written successfully without applying Pair Programming. Pair Programming involves a classical economic tradeoff. When programming in pairs, the personnel cost basically is doubled – that's the cost. On the other hand, first empirical evidence [4, 10, 14] indicates that:

- A pair of programmers has a higher development speed than a single programmer. This is called the *pair speed advantage.*
- The code produced by a pair of programmers has a reduced defect density as compared to the code of a single programmer. This is called the *pair defect advantage.*

The pair speed and defect advantage are the potential benefits of Pair Programming. The question is whether the extra cost of Pair Programming is balanced by the potential benefits.

The potential speed and defect advantage of Pair Programming are usually explained as follows. Pair programming allows developers to share their ideas immediately. This allows to get down to solutions more quickly and also helps to eliminate defects early. In addition, Pair Programming leads to an ongoing review of the program code by the second developer, which reduces the defect density of the code.

In this paper, we construct a mathematical model for the economic value of software development projects. The model can be adapted to both conventional projects and Pair Programming projects by suitably choosing the model parameters. The model focuses on development cost, not on operation cost. We apply the model to a hypothetical, but realistic software project in two different scenarios: the first scenario corresponds to conventional development, the second scenario uses Pair Programming. This way we can compare the economics of Pair Programming with the economics of conventional development. We give a comprehensive and detailed analysis of how efficient Pair Programming must be in order to break even with the conventional process.

Our economic project model is based on the concept of *net present value.* With net present value, the returns of a project are discounted back at a certain rate. The discount rate models the fact that returns which are realized sooner are more valuable than returns which are realized later. The Extreme Programming community considers their special techniques 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 market pressure.

In our study, we systematically vary model parameters for the sample project to see how sensitive the cost-benefit relation of Pair Programming is to changes in the project setting. As expected, we find that the cost-benefit relation depends on how large the pair speed and defect advantage actually are.

We also find that in many cases the pair and defect advantage do not completely determine the cost-benefit relation of Pair Programming: the market pressure, as modelled by the discount rate, plays a decisive role. For many reasonable values of the pair speed and defect advantage, Pair Programming only pays off under strong market pressure. With strong market pressure, the increased personnel cost is balanced by a gain in market share resulting from a shorter time to market. We use the concept of *break-even discount rate* to analyze how strong the market pressure must be for Pair Programming to break even when the other model parameters are kept fixed. The break-even discount rate turns out to be a useful tool for this kind of analysis.

The results of our study have direct implications for software management practice. Our most important findings are as follows:

- Our study indicates that a manager should consider using Pair Programming given that the market pressure is very strong and his programmers are much faster when working in pairs as compared to working alone.
- On the other hand, if the size of the workforce does not allow to run the project with enough pairs to exploit the degree of parallelism possible in the project, a manager should consider adding to the workforce of single programmers instead of using Pair Programming.

As becomes clear from the preceding discussion, it is *not* sufficient to study the pair speed and defect advantage alone in order to understand, model, and evaluate the impact of Pair Programming on software development: *we need a combination of metrics from different categories* to get the full picture.

Besides *process metrics* such as the pair speed advantage, we must take into account *product metrics,* such as the module breakdown structure of the software. The module breakdown structure determines the maximum number of tasks that can be worked on at the same time in the project, which in turn is a limit for adding developers in a conventional project, respectively, pairs in a Pair Programming project. If the task limit has been reached, adding developers to a conventional project in order to *form* pairs still can be beneficial since this might speed up the project due to the pair speed advantage. Note that in this study we treat programmer pairs as entities; hence, the communication overhead in a project which uses Pair Programming is assumed to increase with the number of *pairs* and not with the number of individuals.

In addition, when evaluating Pair Programming we must take into account *project context metrics,* such as the market pressure. Adding developers to form programmer pairs only pays off when the market pressure is high. Finally, the various metrics are *integrated* by our economic model for the business value of a project. The model not only allows to study the cost-benefit relation of Pair Programming, but also the relative strength of the impact of the metrics on the result.

The interplay between the metrics turns out to be interesting and fairly complex.

We have presented and discussed an earlier version of our model at a workshop [9]. The comprehensive break-even analysis for the market pressure which we present here is completely new.

2. Related Work

We are aware of only two empirical studies which provide some quantitative evidence for the benefits of Pair Programming. Both studies indicate that the pair speed advantage (one programmer pair versus a single developer) actually does exist, but the authors come to different numbers. Nosek [10] reports about a study with software professionals where the pairs on average had a 29 percent shorter time to completion than the single programmers. Williams [4, 14] reports about a study with undergraduate students where the pairs required between 20 and 40 percent less time for completing their task than single developers. Williams [4] also reports that pair programming led to 15 percent fewer defects in the final product as compared to single developers. An early paper by Bisant and Lyle [3] already indicated that working in pairs during a review can save total development effort despite having doubled personnel cost during the review.

The results of our previous workshop paper [9] have been independently replicated and confirmed by Smith and Menzies [11]. Smith and Menzies are motivated by the question whether lightweight methods should be adopted by NASA, or not.

Williams and Erdogmus [13] present a different study about the economic feasibility of pair programming which is also based on the concept of net present value. There are a number of major differences between their work and ours, though. In the study of Williams and Erdogmus, Pair Programming is run under a software factory model where code is not only developed, but also delivered *and paid for* in very small increments. This assumption is unrealistic even for Extreme Programming projects, which typically are small scale. Williams and Erdogmus adopt the most optimistic figures about the speed and defect advantage of pairs reported earlier by Williams. In particular, pairs are assumed to work almost twice as fast as individuals. No sensitivity analysis with respect to the pair speed and defect advantage is provided. As opposed to our study, the break-even analysis of Williams and Erdogmus focuses on unit value, that is, the amount of dollars earned per line of code, instead of market pressure.

Erdogmus and Williams conclude that there is an *overall* economic advantage of 40 percent for pairs over single programmers. Due to their modelling approach, this figure is *independent* of the actual discount rate, product size, project deadline, and labor cost. We doubt that such a global figure is valuable as a basis for management decisions in a given project setting. In addition, their result depends on their particular choice of the pair speed and defect advantage.

In contrast, we use a more realistic project model. Our sensitivity analysis shows how strongly the economic value of the Pair Programming project depends on the pair speed advantage and pair defect advantage. Our study also makes clear that even when adopting the most optimistic figures reported in the literature to date about the speed and defect advantage of pairs, market pressure must be rather strong in order for Pair Programming to break even.

3. Input Metrics

In this section, we describe in detail the different metrics that we use as input for our economic project model. Some of the metrics are tailored to Pair Programming, such as the pair speed advantage, others come from software engineering economics and standard economics, such as the product size and the discount rate. The connection between the different metrics will be made clear in the next section.

3.1. Process Metrics

We use the following process metrics for the conventional process and for Pair Programming:

- the productivity of a single developer;
- the pair speed advantage;
- the defect density of code;
- the pair defect advantage;
- the defect removal time.

The average Productivity of a single developer is measured in lines of code per month. Figures in the literature for the average productivity range between 250 and 550 lines of code per month, including design, coding, and unit testing, but excluding regression testing [12].

A central claim of Pair Programming is that a pair of programmers has a much higher development speed than a single programmer. To measure the difference in development speed, we use a process metric which is

tailored to Pair Programming: the pair speed advantage. The PairSpeedAdvantage is defined as the ratio between the time required by a single developer and the time required by a pair of programmers for some given task.

For example, Nosek [10] reports that programmer pairs on average require a 29 percent shorter time to completion for their tasks than single programmers. For this data, we have

$$\mathsf{PairSpeedAdvantage} = \frac{100}{100 - 29} = 1.4.$$

From the few existing empirical studies we know that the pair speed advantage can reasonably be expected to range between 1.3 and 1.8 [4, 10, 14].

A programmer typically inserts 100 defects per thousand lines of code [8]. A good conventional software process eliminates up to 70 percent of these defects [8]. Therefore, the code produced with conventional development is assumed to have an average defect density of

$$\mathsf{DefectDensity} = \frac{100}{1000} \times \frac{30}{100} = 0.03$$

defects per line of code.

Pair Programming claims to lead to fewer defects in the code as compared to conventional development. To measure the difference in code quality, we use another process metric which is tailored to Pair Programming: the pair defect advantage. The PairDefectAdvantage is defined as 100 percent minus the ratio between the defect density of the Pair Programming process and the defect density of the conventional process. The existing empirical studies indicate that the PairDefectAdvantage ranges about 15 percent; that is, Pair Programming on average leaves 15 percent fewer defects in the code than conventional development [4, 14].

The time needed to remove a defect in quality assurance is denoted as DefectRemovalTime. Figures in the literature for the defect removal time vary between 5 and 20 hours per defect [7, 8].

3.2. Product Metrics

We use the following product metrics of the software to be developed as input:

- the product size;
- the module breakdown structure of the software.

The ProductSize is measured in lines of code. The product size is the same for conventional development and Pair Programming.

The module breakdown structure of the software determines the maximum number of tasks that can reasonably be worked on simultaneously in the project. Splitting tasks any further doesn't make sense due to the size and structure of the software. The maximum number of tasks is denoted by TaskLimit. The maximum number of tasks is an upper limit for adding developers to a conventional project. Instead of measuring the module breakdown structure in detail, it suffices for our purposes to directly use the metric TaskLimit as input to our economic model.

3.3. Project Context Metrics

The business value of a software project depends on a number of factors from the economic context of the project. We use the following project context metrics as input:

- the discount rate;
- the asset value;
- the number of single developers;
- the number of programmer pairs;
- the developer salary;
- the project leader salary;
- the monthly working hours.

We use the DiscountRate as a measure for the market pressure, see the next section. In order to model strong market pressure, we use discount rates between 25 and 100 percent a year. Such a high discount rate means that time to market is a decisive factor for the business value of the project.

The AssetValue is the amount of dollars returned by the customer once the software is complete and operational.

The number of single developers in the conventional project is denoted by NumOfDevelopers. The number of programmer pairs in the Pair Programming project is denoted by NumOfPairs. For some computations, we'll assume that the number of pairs equals half the number of single developers; for other computations, we'll assume that workforce has been added to form additional programmer pairs.

We'll assume that the DeveloperSalary is $50,000$ dollars per year and the project LeaderSalary $60,000$ dollars per year. A reasonable figure for the monthly working hours WorkTime of a developer is 135 hours.

4. Economic Model

In this section, we describe in detail our model for the economic value of a software development project. The model can be applied to both conventional projects and Pair Programming projects by suitably choosing the values of the model parameters. The model takes the metrics which we have described in the preceding section as input.

4.1. Net Present Value

Our model for the economic value of a development project is based on the concept of *net present value.* The net present value of a project is defined as [5, 6]

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

With net present value, the dollar returns of a project (AssetValue) are discounted at a certain rate, the DiscountRate. The rationale behind discounting is that an investment worth one dollar today is worth

$$(\,1 \;+\; \mathsf{DiscountRate}\,)^{\,T}$$

dollars in T periods. With this rationale, the *present* value of the project must be calculated by discounting *back* the asset value from the time of project completion (DevTime) to time zero, and then deducing the development cost (DevCost). A project has business value only if its net present value is positive. Otherwise, the project leads to a financial loss.

Time to market can be the decisive factor for the success of a project. For such a project, a delay of the project's completion leads to a loss of market share, which drastically decreases the business value of the whole project. To take strong market pressure into account, it is common in economics to choose high values for the discount rate in the formula for the net present value.

Factors such as the development time, development cost, and net present value of the project are some sort of "output" metrics which are derived from the input metrics described in the preceding section.

4.2. Development Time

For conventional projects, the development time (measured in years) is calculated as

$$\mathsf{DevTime}_{\mathsf{C}} \;=\; \frac{1}{12} \times \frac{\mathsf{ProductSize}}{\mathsf{Productivity} \times \mathsf{NumOfDevelopers}} \;+\; \mathsf{QATime}.$$

Since the Productivity is measured in lines of code per month instead of per year, we have to divide by 12 to get the unit right.

The time needed for *additional* quality assurance (QATime) is special: it's the time needed to *compensate* the defect advantage which Pair Programming is assumed to have over the conventional process. The quality assurance time is computed in the next subsection.

For a project which uses Pair Programming, no additional time for quality assurance is required ($\mathsf{QATime} = 0$), but the fact that developers work in pairs must be taken into account:

$$\mathsf{DevTime}_{\mathsf{PP}} \;=\; \frac{1}{12} \times \frac{\mathsf{ProductSize}}{\mathsf{Productivity} \times \mathsf{NumOfPairs}} \times \frac{1}{\mathsf{PairSpeedAdvantage}}.$$

In particular, the pair speed advantage enters the formula for the development time of the Pair Programming project.

In our model, we 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.

4.3. Quality Assurance

Using conventional development, there are

$$\mathsf{DefectsLeft} \;=\; \mathsf{ProductSize} \times \mathsf{DefectDensity}$$

defects left in the software after coding. Pair Programming claims to produce code which has a reduced defect density. The difference in code quality is measured by the pair defect advantage. 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 Pair Programming, no separate quality assurance phase is required.

The length of the separate quality assurance phase for the conventional process depends on the defect

difference and the average time needed to remove a single defect:

$$\mathsf{QATime} = \frac{1}{12} \times \frac{\mathsf{DefectRemovalTime}}{\mathsf{WorkTime} \times \mathsf{NumOfDevelopers}} \times \mathsf{DefectDifference}.$$

4.4. Development Cost

For simplicity, our model assumes that the development cost of a project only consists of the salaries for the developers and the project leader. The model does not take into account project startup cost, nor product installation cost.

For the cost of the conventional project, we get:

$$\mathsf{DevCost_C} = \mathsf{DevTime_C} \times (\mathsf{NumOfDevelopers} \times \mathsf{DeveloperSalary} + \mathsf{LeaderSalary}).$$

For the Pair Programming project, we get:

$$\mathsf{DevCost_{PP}} = \mathsf{DevTime_{PP}} \times (2 \times \mathsf{NumOfPairs} \times \mathsf{DeveloperSalary} + \mathsf{LeaderSalary}).$$

5. Numerical Results

In this section, we compute the net present value of a hypothetical, but realistic sample project for various project settings. We distinguish between two main scenarios: In the first scenario, the project is run using a conventional process. In the second scenario, the project is run using Pair Programming. For different values of the pair speed advantage, pair defect advantage, and discount rate we compare the net present value $\mathsf{NPV_{PP}}$ of the Pair Programming project against the net present value $\mathsf{NPV_C}$ of the conventional project.

5.1. Sample Project

We keep some model parameters fixed for the sample project, see TABLE 1.

To get some impression how large the sample project actually is, assume that we have eight developers who follow a conventional process. In this case, the formula given in the preceding section for the development time of conventional projects tells us that it would take about half a year to finish the project. In addition, if we assume a moderate annual discount rate of 10 percent, the formula for the net present value of conventional projects yields

$$\mathsf{NPV_C} = 723,463 \text{ dollars}.$$

Table 1. Fixed model parameters for sample project.

parameter	value
Productivity	350 LOC/month
DefectDensity	0.03 defects/LOC
DefectRemovalTime	10 hours/defect
ProductSize	16,800 LOC
TaskLimit	8 tasks
AssetValue	1,000,000 dollars
DeveloperSalary	50,000 dollars/year
LeaderSalary	60,000 dollars/year
WorkTime	135 hours/month

5.2. Strong Market Pressure

Suppose that the sample project is subject to strong market pressure. Also assume that there is a fairly large pool of developers available who could work on the project. Since the number of tasks which can reasonably be worked on simultaneously is bounded by eight ($\mathsf{TaskLimit} = 8$) for this project, the manager has two options:

- He could run the project with up to eight single developers.
- He could run the project with up to sixteen developers who work in pairs.

Clearly, with Pair Programming the personnel cost of the project basically would double. On the other hand, granted that pairs have a speed advantage over single developers, the Pair Programming project should deliver faster than the conventional project. Under strong market pressure, earlier time to market will result in a gain in market share. Thus, the increased personnel cost of Pair Programming should be more than covered for by the gain in market share.

To make a decision which way the project should go, we apply our economic model assuming a high annual discount rate of 75 percent, which corresponds to strong market pressure. We then compare the conventional project with the maximum workforce of eight single developers against Pair Programming with the

maximum workforce of eight pairs, using different values of the pair speed advantage PSA and the pair defect advantage PDA, see TABLE 2.

Table 2. Net present value of sample project under strong market pressure.

PSA	PDA	NPV_C	NPV_{PP}
1.4	5 %	508,803	511,700
1.4	25 %	441,177	511,700
1.8	5 %	508,803	617,141
1.8	25 %	441,177	617,141

TABLE 2 leads to the following conclusions. Given that the speed advantage of pairs is significant – say, equal to Nosek's value of 1.4 – Pair Programming outperforms the conventional project if the pair defect advantage is not too small. If the pair speed advantage is large – say, equal to Williams' value of 1.8 – Pair Programming outperforms the conventional project even when the pair defect advantage is small.

As a management guideline, a manager should consider using Pair Programming if the market pressure is really strong and his programmers are much faster when working in pairs as compared to working alone, given that there is a sufficiently large workforce available. For high discount rates, the speed and defect advantage of pairs come into full play.

5.3. Limited Workforce

Suppose that the workforce available for the project is strictly limited to eight developers. The manager then has two options:

- He could run the project with eight single programmers.
- He could run the project with four programmer pairs.

Even when assuming that pairs have a considerable speed advantage over single developers, it is doubtful whether the speed advantage suffices to compensate the fact that four pairs do not exploit the maximum degree of parallelism possible in the project.

To study the setting with a limited workforce, we apply our economic model assuming a moderate annual discount rate of 25 percent. We then compare the conventional project with eight single developers against Pair Programming with four pairs, using different values of the pair speed advantage PSA and the pair defect advantage PDA, see TABLE 3.

Table 3. Net present value of sample project with limited workforce.

PSA	PDA	NPV_C	NPV_{PP}
1.4	15 %	626,026	524,093
1.8	15 %	626,026	627,851
1.8	25 %	600,509	627,851

TABLE 3 shows that for reasonable values of the pair speed advantage and pair defect advantage the net present value of the conventional project will exceed the net present value of the Pair Programming project. Recall that a value for the pair speed advantage of 1.8 means that pairs work almost twice as fast as single developers, and we have only very limited empirical evidence to date in favor of such a large advantage.

The picture does not change much when the market pressure is strong. Even for a discount rate of 75 percent, a large pair speed advantage of 1.8, and a significant pair defect advantage of 15 percent, the Pair Programming project just breaks even with the conventional project given that the workforce is limited to eight developers.

As a management guideline, a manager should add to the workforce of single programmers to maximize the degree of parallelism in the project instead of using Pair Programming if the size of the workforce does not allow to run the project with the maximum number of pairs. This holds in particular if the market pressure is only moderate. In such a setting, the speed and defect advantage of pairs do not compensate the lack of parallelism in the Pair Programming project.

6. Market Pressure Analysis

Last section's computations for the sample project have shown that adding developers to a project in order to form programmer pairs can be rewarding if the market pressure is high. Thus, it is natural to ask:

> How strong must the market pressure be for the Pair Programming project to break even with the conventional project?

Certainly, the answer depends on the values of other model parameters such as the pair speed advantage.

In this section, we introduce the concept of break-even discount rate and analyze for our sample project how the break-even point changes when important model parameters vary.

We always compare the Pair Programming project – with a possibly varying number of pairs – against the conventional project with eight developers. We also use the model parameters specified in TABLE 1.

6.1. Break-Even Discount Rate

The discount rate for which the net present value of the Pair Programming project breaks even with the net present value of the conventional project is called the *break-even discount rate.* More formally, the break-even discount rate is the solution to the equation

$$\mathsf{NPV_{PP}}(\mathsf{DiscountRate}) = \mathsf{NPV_C}(\mathsf{DiscountRate}).$$

Here, we view the net present value as a function with the discount rate as the independent variable.

The break-even discount rate depends on the values of the other model parameters. In FIGURE 1, we systematically vary the pair speed advantage and the pair defect advantage for the sample project while retaining the other model parameters. The number of pairs is equal to 6. We have cut off the surface in FIGURE 1 at a z-level of 200 percent.

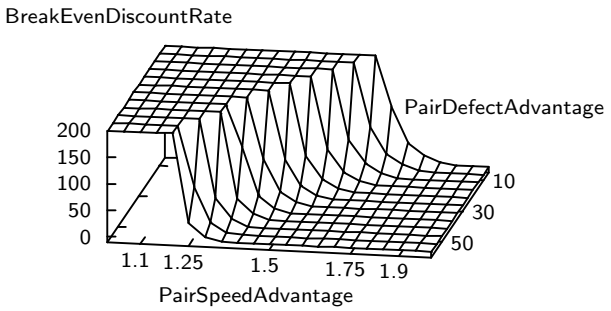

Figure 1. Break-even discount rate dependent on the pair speed advantage and pair defect advantage.

As could be expected, for a large pair speed advantage or a large pair defect advantage Pair Programming can break even with the conventional project already for moderate market pressure. On the other hand, for small values of the pair speed advantage and pair defect advantage, high or even excessive market pressure is required for Pair Programming to break even.

6.2. Impact of Pair Speed Advantage

It is worthwhile to study the impact of the pair speed advantage on the break-even discount rate when all other model parameters are fixed, including the pair defect advantage. For example, if we fix the pair defect advantage at 15 percent and compute the break-even discount rate while varying the pair speed advantage in steps of 0.01, we get the curve[1] in FIGURE 2.

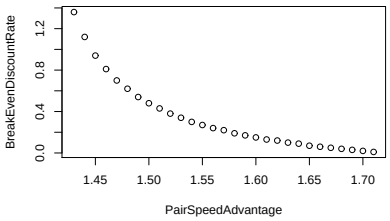

Figure 2. Break-even discount rate dependent on the pair speed advantage.

This curve looks much like an exponential function. Indeed, if we take log values and perform a linear regression, we get an almost perfect fit when we leave out those points where the break-even discount rate is below 5 percent, see FIGURE 3.

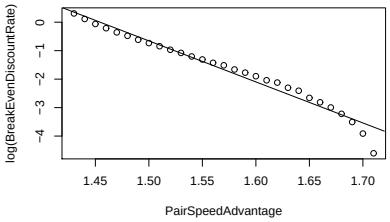

Figure 3. Logarithm of break-even discount rate depends linearly on the pair speed advantage.

It is much easier to compare the slopes and intercepts of straight lines than to compare non-linear curves. The fact that we can view the logarithm of the break-even discount rate as a *linear* function of the pair speed advantage simplifies the sensitivity analysis for the break-even discount rate. Note that this (approximate) log-linear relationship is not obvious from the model equations.

For our present study, it makes sense to disregard values of the discount rate below 5 percent, since small discount rates do not correspond to market *pressure.*

[1] This curve is the intersection of the surface in FIGURE 1 with the hyperplane $\mathsf{PairDefectAdvantage} = 15$.

The slight deviation of some of the points from the regression line has no impact on our results.

6.3. Impact of Pair Defect Advantage

To study the impact of the pair defect advantage on the break-even discount rate, we take advantage of the (approximate) log-linear relationship between the pair speed advantage and the break-even discount rate. For example, FIGURE 4 shows the different regression lines that we get when the pair defect advantage varies between 5 and 25 percent. The number of pairs is assumed to be 6. The picture looks similar for other values for the numbers of pairs.

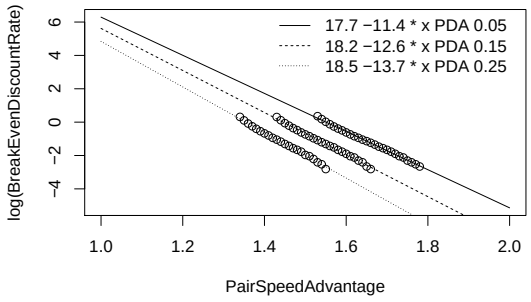

Figure 4. Regression lines for different values of the pair defect advantage.

The regression lines for small values of the pair defect advantage lie above the regression lines for large values of the pair defect advantage. In addition, the slope of the regression line increases as the pair defect advantage increases.

FIGURE 4 provides interesting conclusions:

- The larger the pair defect advantage, the smaller the pair speed advantage and discount rate which are required for Pair Programming to break even. This follows from the relative position of the regression lines.
- The impact of the pair speed advantage on the break-even discount rate is stronger for large values of the pair defect advantage. This follows from the slopes of the regression lines.

TABLE 4 lists for different values of the pair defect advantage the range of the pair speed advantage where the break-even discount rate lies between 5 and 150 percent. The number of pairs is assumed to be 6. One can see how the relevant range shifts to the left as the pair defect advantage increases.

Table 4. Relevant range of pair speed advantage for different pair defect advantages.

PairDefectAdvantage	PairSpeedAdvantage
5	1.53 ... 1.78
10	1.48 ... 1.72
15	1.43 ... 1.66
20	1.38 ... 1.60
25	1.34 ... 1.55
30	1.30 ... 1.50
35	1.26 ... 1.46
40	1.23 ... 1.41

6.4. Impact of the Number of Pairs

Another important model parameter is the number of pairs in the Pair Programming project. Recall that the number of developers in the conventional project is fixed at eight, but the number of pairs in the Pair Programming project may vary.

Again, we can exploit the linear relationship between the logarithm of the break-even discount rate and the pair speed advantage to study the impact of the number of pairs. FIGURE 5 shows the different regression lines we get when the number of pairs varies from five to eight. The pair defect advantage is fixed at 15 percent. The picture looks similar for other values of the pair defect advantage.

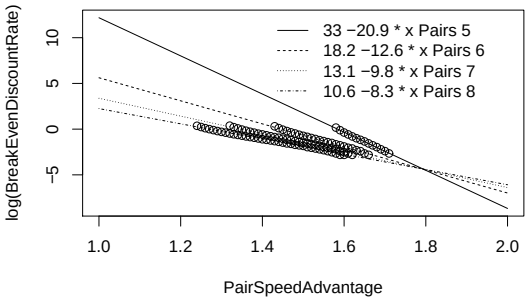

Figure 5. Regression lines for varying number of pairs.

The regression lines for small numbers of pairs lie above the lines for large numbers of pairs. The slope of

the regression line decreases as the number of pairs increases. The intersection point of the lines in FIGURE 5 must be disregarded, since it occurs in an area where the break-even discount rate drops below 5 percent.

Again, we can draw interesting conclusions from FIGURE 5:

- The larger the workforce of programmer pairs, the smaller the pair speed advantage and discount rate which are required to break even.
- The impact of the pair speed advantage on the break-even discount rate is stronger for small numbers of pairs.

TABLE 5 lists for different numbers of pairs the range of the pair speed advantage where the break-even discount rate lies between 5 and 150 percent. The pair defect advantage is assumed to be 15 percent. One can see how the relevant range shifts to the left as the number of pairs increases.

Table 5. Relevant range of pair speed advantage for varying number of pairs.

NumOfPairs	PairSpeedAdvantage
4	1.80 ... 2.00
5	1.58 ... 1.71
6	1.43 ... 1.66
7	1.32 ... 1.62
8	1.24 ... 1.60

6.5. Impact of Other Parameters

We have repeated our computations using different values for other model parameters such as the average productivity of a developer or the defect removal time. Although the numbers change, the overall picture remains the same.

For example, we have systematically varied the productivity from 250 to 450 lines of code per month. FIGURE 6 shows the corresponding regression lines. The pair defect advantage is fixed at 15 percent and the number of pairs at 6.

The higher the productivity, the lower the break-even discount rate for a given pair speed advantage. Since the regression lines for the different productivity values have a similar slope, the strength of the impact of the pair speed advantage on the break-even discount rate is by and large independent of the value for the productivity.

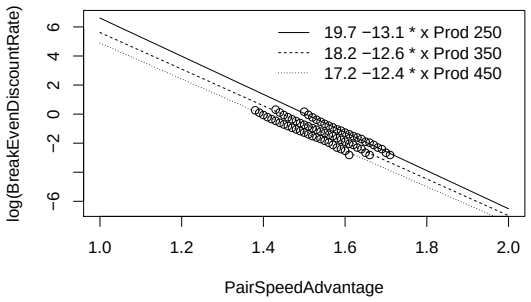

Figure 6. Regression lines for varying developer productivity.

FIGURE 7 shows the regression lines we get when we vary the defect removal time from 5 to 20 hours. The pair defect advantage is fixed at 15 percent, the productivity at 350 lines of code per month, and the number of pairs at 6.

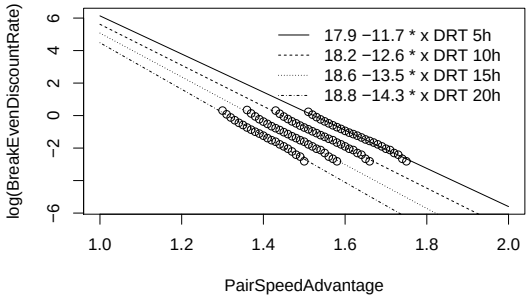

Figure 7. Regression lines for varying defect removal time.

Recall that the defect removal time has a negative impact on the net present value of the conventional project only if the pair defect advantage is non-zero. The longer the time needed to eliminate a defect, the more time the conventional project must spend in the separate quality assurance phase to catch up with the higher quality code of the Pair Programming project. Thus, the potential benefit of Pair Programming increases with the defect removal time. In particular, the impact of the pair speed advantage on the break-even discount rate increases with the defect removal time, see the slope of the regression lines.

6.6. Alternative Approach

Instead of studying the relationship between the break-even discount rate and the pair speed advantage, one might as well fix the pair speed advantage and study the break-even discount rate as a function of the pair defect advantage. Again, it turns out that the logarithm of the break-even discount rate can be approximately viewed as a *linear* function of the pair defect advantage. FIGURE 8 shows the corresponding regression lines for different values of the pair speed advantage. The number of pairs was set to 6. As before, points where the break-even discount rate falls below 5 percent were left out.

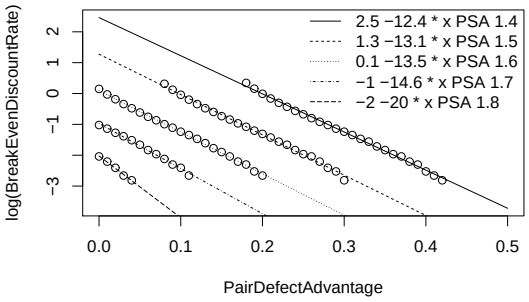

Figure 8. Logarithm of break-even discount rate depends linearly on the pair defect advantage.

One could repeat the break-even analysis given in the preceding subsections using the pair defect advantage as the independent variable instead of the pair speed advantage. The results remain the same, but one can gain additional insight into the dependencies among the model parameters. For example, FIGURE 9 makes the strong relationship between the pair defect advantage and the defect removal time apparent. The pair speed advantage has been set to 1.6 and the number of pairs is equal to 6.

7. Conclusions

We have shown how to combine different metrics to evaluate the cost and benefit of Pair Programming. We have integrated process metrics, product metrics, and process context metrics into a model for the business value of a project. The model is based on the concept of net present value, which allows to consider the impact of time to market on the value of a project.

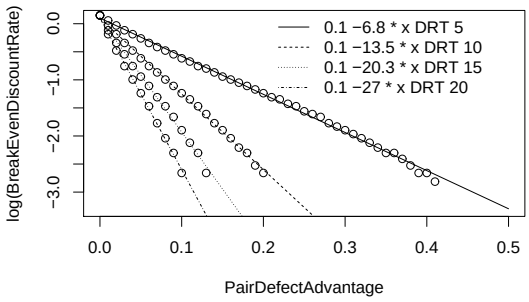

Figure 9. Relationship between pair defect advantage and defect removal time.

We have applied our model to different project scenarios and provided a detailed sensitivity analysis with respect to the model parameters. The pair speed advantage, pair defect advantage, discount rate, and number of pairs each have a strong impact on the value of a project which uses Pair Programming.

The results of our computations provide clear management guidelines when to use Pair Programming or better not. Given that a short time to market is decisive for the success of a project, adding developers to form programmer pairs can speed up the project and increase its business value despite the increased personnel cost. This is due to the fact that programmer pairs can be expected to have a higher productivity and code quality as compared to single programmers.

We feel that it is valuable to see the complex interplay between the different metrics and to understand that the software process metrics tailored to Pair Programming alone do not suffice to properly assess the value of Pair Programming. We consider techniques from economics to be the right vehicle to combine software engineering metrics in order to assess the tradeoffs involved in a new development paradigm such as Pair Programming.

One important task for future research about Pair Programming is to collect reliable empirical data about how large the pair speed and defect advantage actually are; currently, empirical evidence is very limited.

References

[1] K. Beck. Embracing change with extreme programming. *IEEE Computer*, pages 70–77, Oct. 1999.

[2] K. Beck. *Extreme Programming Explained*. Addison Wesley, 1999.

[3] D. Bisant and J. Lyle. A two-person inspection method to improve programming productivity. *IEEE Transactions on Software Engineering*, 15(10):1294–1304, Oct. 1989.

[4] 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.

[5] H. Erdogmus. Comparative evaluation of software development strategies based on net present value. In *International Workshop on Economics-Driven Software Engineering Research* EDSER, Los Angeles, USA, May 1999.

[6] W. Harrison, D. Raffo, and J. Settle. Measuring the value from improved predictions of software process improvement outcomes using risk-based discount rates. In *International Workshop on Economics-Driven Software Engineering Research* EDSER, Los Angeles, USA, May 1999.

[7] W. Humphrey. *Managing the Software Process*. Addison-Wesley, 1989.

[8] W. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, 1997.

[9] M. M. Müller and F. Padberg. Extreme programming from an engineering economics point of view. In *International Workshop on Economics-Driven Software Engineering Research* EDSER, Orlando, Florida, May 2002.

[10] J. Nosek. The case for collaborative programming. *Communications of the ACM*, 41(3):105–108, Mar. 1998.

[11] J. Smith and T. Menzies. Should NASA embrace agile processes, 2002. preprint, West Virginia University, Morgantown, USA.

[12] I. Sommerville. *Software Engineering*. Addison-Wesley, 1996.

[13] L. Williams and H. Erdogmus. On the economic feasibility of pair programming. In *International Workshop on Economics-Driven Software Engineering Research* EDSER, Orlando, Florida, May 2002.

[14] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair-programming. *IEEE Software*, pages 19–25, July/Aug. 2000.