On the Economic Evaluation of XP Projects

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

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

ABSTRACT

From a project economics point of view, the most important practices of Extreme Programming (XP) are Pair Programming and Test-Driven Development. Pair Programming leads to a large increase in the personnel cost, and Test-Driven Development adds to the development effort. On the other hand, Pair Programming can speed the project up; both Pair Programming and Test-Driven Development can reduce the defect density of the code. Can the increased cost of XP be balanced by its shorter time to market and higher code quality?

To answer this question, we construct a new model for the business value of software projects. We then analyze the cost and benefit of XP by applying our model to a realistic sample project. We systematically vary important model parameters to provide a sensitivity analysis. Our analysis shows that the economic value of XP strongly depends on how large the XP speed and defect advantage really are. We also find that the market pressure is an important factor when assessing the business value of XP. Our study provides clear guidelines for managers when to consider using XP – or better not.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics, Management

General Terms

Economics, Management, Measurement

Keywords

Extreme Programming, Cost-Benefit Analysis

1. INTRODUCTION

Lightweight development paradigms such as Extreme Programming (XP) are highly controversial. Proponents of XP claim that their 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. To properly judge a new paradigm such as XP one must study for which project settings the potential benefits of XP balance its cost, if at all. Such a study must be based on objective measures from economics.

In this paper, we construct a model for the economic value of software development projects. We then analyze the cost and benefit of XP by applying our economic model to a hypothetical, yet realistic sample project in two different scenarios: the first scenario corresponds to conventional development, the second scenario uses XP. The team size, product size, and time scale of the sample project are typical for a project where one would consider using XP. In addition, we systematically vary the key model parameters to see how sensitive the results are to a change in the project setting. We arrive at a comprehensive and detailed analysis of how efficient the special practices of XP must be in order for XP to outperform 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. In the XP community, the special practices 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 to explicitly take into account the market pressure.

Extreme Programming breaks with many traditional software engineering practices [1, 2]. From the economics point of view, the most important practices of XP are *Pair Programming, Test-Driven Development,* and *Small Releases.* When running a project with XP, all programming tasks must be performed by *pairs* of programmers using *one* display, keyboard, and mouse. Executable test cases are written *ahead* of the code ("test-first") and serve as a substitute for the specification. The test cases must be re-run continuously during development. There is no separate design or testing phase. Design, implementation, and testing go together in small increments. The formulas in our economic project model directly reflect these practices of XP.

ESEC/FSE'03, September 1–5, 2003, Helsinki, Finland.

Due to Pair Programming, the personnel cost basically is doubled with XP. In addition, Test-Driven Development can lead to a delay in the task completions due to the extra effort required for continuously adapting and running the test cases. The main claim of XP is that this increased cost is more than compensated 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 density.

The potential speed and defect advantage of XP 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. Finally, continuous testing also helps to eliminate defects early.

The potential benefits of XP do not always balance its increased cost. Our sensitivity analysis for the sample project shows that the economic value of XP strongly depends on how large the XP speed and defect advantage actually are, as expected. We also find that the market pressure is an important factor when assessing the business value of a XP project. The results of our study have direct implications for software management practice. Our most important findings are as follows:

- A manager should consider XP if the market pressure is strong and his programmers are much more efficient (with respect to both productivity and code quality) when working in pairs as compared to working alone.
- On the other hand, if the market pressure is only moderate and there are tasks left over which can easily be assigned to additional developers, a manager should add single developers instead of using XP.

These are general guidelines; the decision whether to use XP or not from an economics perspective must be made individually for each project setting. In summary, we get a largely diversified picture of the tradeoff between the cost and benefit of XP.

2. RELATED WORK

The economic project model presented here is based on our workshop paper about Pair Programming [11]. The results of that workshop paper have been independently replicated and confirmed by Smith and Menzies [14]. Smith and Menzies are motivated by the question whether XP (or other lightweight methods) should be adopted by NASA, or not. We have also included results from our workshop paper about Test-Driven Development [12].

Currently, the impact of Pair Programming and Test-Driven Development on the productivity and code quality are only being studied *separately.* For both practices, the empirical knowledge is very limited.

Some empirical studies provide evidence for the benefits of Pair Programming. The studies indicate that the pair speed advantage actually does exist, but the authors come to different numbers. Nosek [13] reports about a study with software professionals where the pairs had a 29 percent shorter time to completion than the individual programmers. Williams [4, 17] reports about a study with undergraduate students where the pairs required between 20 and 40 percent less time for completing their task than the individuals. Williams [4] also reports that Pair Programming led to 15 percent fewer defects in the final product as compared to single programmers. 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.

Test-Driven Development has been studied empirically by one of the authors in [10]. This study provides some evidence that the code quality improves, but the development slows down with Test-Driven Development.

Williams and Erdogmus [16] present another study about the economic feasibility of Pair Programming (without Test-Driven Development) 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 XP 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.

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 study makes clear that even when adopting the most optimistic figures reported in the literature to date about the speed and defect advantage of pairs, the market pressure must be rather strong in order for XP to pay off.

3. 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 XP projects by suitably choosing the values for the model parameters.

3.1 Project 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 [6, 7] is defined as:

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

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

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

dollars in T periods. With this rational, 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.

3.2 Market Pressure

Time to market can be the decisive factor for the success of a project. Under strong market presure, a delay of the project completion leads to a loss of market share, which drastically decreases the business value of the project.

To model strong market pressure, it is common in economics to choose high values for the discount rate in the formula for the net present value. In our study, we use discount rates up to 75 percent a year. A high discount rate means that time to market is a decisive factor for the business value of the development project.

3.3 XP Speed Factor

The key XP practices of Pair Programming and Test-Driven Development have an opposite impact on the task completion times in a project. On the one hand, one can expect that a pair of programmers has a higher development speed than a single programmer; this is supported by first empirical studies [4, 13, 17]. On the other hand, test-driven development seems to lead to a delay in the task completion times due to the extra effort spent on continuously adapting and running the test cases [10].

It is a central claim of XP that the speed advantage predominates, but currently we lack empirical data to confirm this claim. To model the difference in development speed between a conventional project and a XP project in an unbiased way, we introduce the *XP speed factor*. The XPSpeedFactor is defined as the (average) ratio between the time required for some task by a single developer in a conventional project and the time required by a pair of programmers using test-driven development in a XP project. If the XPSpeedFactor is larger than one, XP has a speed advantage over conventional development; otherwise, XP has a speed disadvantage.

The difference in development speed between a conventional project and a Pair Programming project (*without* test-driven development) is measured by the *pair speed advantage*. The PairSpeedAdvantage is defined analogous to the XP speed factor. Nosek [13] reports that programmer pairs on average require a 29 percent shorter time to completion for their tasks than single programmers. Using this data, we have

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

The few empirical studies available today indicate that the pair speed advantage can reasonably be expected to range between 1.3 and 1.8 [4, 13, 17]. The pair speed advantage is an upper bound for the XP speed factor:

$$\mathsf{XPSpeedFactor} \;\leq\; \mathsf{PairSpeedAdvantage}.$$

Similarly, the XP speed factor is bounded from below by the *test-driven speed factor*:

$$\mathsf{TestDrivenSpeedFactor} \;\leq\; \mathsf{XPSpeedFactor}.$$

The test-driven speed factor is defined analogous to the XP speed factor but relates to test-driven development (without Pair Programming). In principle, the range for the test-driven speed factor can be determined empirically in the same way as the pair speed advantage.

3.4 XP Defect Factor

A programmer typically inserts 100 defects per thousand lines of code [9]. A good conventional software process eliminates up to 70 percent of these defects [9]. Therefore, we assume that the code produced with conventional development has an average defect density of

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

defects per line of code.

XP claims that Pair Programming and Test-Driven Development lead to fewer defects in the code as compared to conventional development. To model the difference in code quality between a conventional project and a XP project in an unbiased way, we introduce the *XP defect factor*:

$$\mathsf{XPDefectFactor} \;=\; \frac{\mathsf{DefectDensity}_{\mathsf{C}} \;-\; \mathsf{DefectDensity}_{\mathsf{XP}}}{\mathsf{DefectDensity}_{\mathsf{C}}}.$$

$\mathsf{DefectDensity}_{\mathsf{XP}}$ denotes the (average) defect density of a XP project.

Again, we currently lack empirical data about how large the XP defect factor actually is. One empirical study indicates though that Pair Programming (*without* test-driven development) already provides a *pair defect advantage* over conventional development which ranges about 15 percent; that is, Pair Programming on average leaves 15 percent fewer defects in the code than conventional development [4, 17]. Although empirical evidence is limited yet, we expect the XP defect factor to actually measure an advantage since the defect advantage of Pair Programming should be reinforced by test-driven development. Therefore, the XP defect factor is bounded from below by the pair defect advantage:

$$\mathsf{PairDefectAdvantage} \;\leq\; \mathsf{XPDefectFactor}.$$

3.5 Workforce Level

The number of single developers in the conventional project is denoted by NumOfDevelopers. The number of programmer pairs in the XP 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 more programmer pairs.

The *module breakdown structure* of the software determines the maximum number of tasks that can be worked on simultaneously in the project. Splitting the development 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, which 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 for our computations.

3.6 Development Time

In our economic model, we compute the development time for conventional projects as

$$\begin{aligned}\mathsf{DevTime}_{\mathsf{C}} &= \frac{1}{12} \times \frac{\mathsf{ProductSize}}{\mathsf{Productivity} \times \mathsf{NumOfDevelopers}} \\ &+ \mathsf{QATime}.\end{aligned}$$

The ProductSize is measured in lines of code. The product size is the same for conventional development and XP. 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 [15].

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

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

$$\begin{aligned}\mathsf{DevTime}_{\mathsf{XP}} &= \frac{1}{12} \times \frac{\mathsf{ProductSize}}{\mathsf{Productivity} \times \mathsf{NumOfPairs}} \\ &\times \frac{1}{\mathsf{XPSpeedFactor}}.\end{aligned}$$

In addition, the XP speed factor enters the formula for the development time of the XP 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.

3.7 Quality Assurance

Using conventional development, there are

$$\mathsf{DefectsLeft} = \mathsf{ProductSize} \times \mathsf{DefectDensity}_{\mathsf{C}}$$

defects left in the software after coding. Given that XP produces code which has a reduced defect density, the conventional project must make up for the quality difference of

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

defects in a separate quality assurance phase before entering the market. With XP, no separate quality assurance phase is required.

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 15 hours per defect [8, 9]. The length of the separate quality assurance phase for the conventional process depends on the defect difference and the defect removal time:

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

A reasonable figure for the monthly working hours of a developer (WorkTime) is 135 hours.

3.8 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.

For the cost of the conventional project, we get:

$$\begin{aligned}\mathsf{DevCost}_{\mathsf{C}} &= \mathsf{DevTime}_{\mathsf{C}} \\ &\times (\mathsf{NumOfDevelopers} \times \mathsf{DeveloperSalary} \\ &+ \mathsf{LeaderSalary}).\end{aligned}$$

For the XP project, we get:

$$\begin{aligned}\mathsf{DevCost}_{\mathsf{XP}} &= \mathsf{DevTime}_{\mathsf{XP}} \\ &\times (2 \times \mathsf{NumOfPairs} \times \mathsf{DeveloperSalary} \\ &+ \mathsf{LeaderSalary}).\end{aligned}$$

The model does not take into account project startup and fixed cost, such as the salary for secretaries or the cost for computer equipment, nor does the model take into account product installation cost. It is reasonble to assume that startup cost, fixed cost, and installation cost are independent of the development method; thus, these costs can be left out when comparing the net present values of the conventional and XP project.

4. 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 XP. For different values of the XP speed factor, XP defect factor, and discount rate we compare the net present value of the XP project against the net present value of the conventional project. More precisely, we study the ratio

$$\frac{\mathsf{NPV_{XP}} - \mathsf{NPV_C}}{\mathsf{NPV_C}}$$

where $\mathsf{NPV_{XP}}$ denotes the net present value of the XP project and $\mathsf{NPV_C}$ denotes the net present value of the conventional project. The NPV ratio captures the relative advantage (or disadvantage) of XP over the conventional process.

4.1 Sample Project

For the sample project that we shall study in the remainder of this paper, we keep some model parameters fixed, see Table 1.

Table 1: Fixed model parameters for sample project.

parameter	value
Productivity	350 LOC/month
DefectDensity (conv.)	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

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 yields that it will 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

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

4.2 Limited Workforce

The number of tasks which can reasonably be worked on simultaneously in the sample project is bounded by eight ($\mathsf{TaskLimit} = 8$). 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 using a conventional process.
- He could run the project with four programmer pairs using XP.

Even when assuming that pairs have a considerable speed advantage over single developers, it is questionable 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 this setting of a limited workforce, we apply our economic model first assuming a moderate annual discount rate ($\mathsf{DiscountRate} = 25\,\%$). We then compare the conventional project with eight single developers against XP with four pairs, using different values for the XP speed factor XPSF and the XP defect factor XPDF, see Table 2.

Table 2: Net present value of sample project with limited workforce under moderate market pressure.

XPSF	XPDF	$\mathsf{NPV_C}$	$\mathsf{NPV_{XP}}$
1.4	15 %	626,026	524,093
1.8	15 %	626,026	627,851
1.8	25 %	600,509	627,851

Table 2 shows that for reasonable values of the speed and defect factor the net present value of the conventional project will exceed the net present value of the XP project. Recall that a value for the XP speed factor of 1.8 means that pairs work almost twice as fast as single developers despite the additional effort required by test-driven development, and we have only very limited empirical evidence to date in favor of such a large speed advantage.

The NPV ratios do not change much when the market pressure is strong. Even for an extreme DiscountRate of 75 percent, a large XP speed factor of 1.8, and a significant XP defect factor of 15 percent, the XP project just breaks even with the conventional project given that the workforce is limited to eight developers, see Table 3.

Table 3: Net present value of sample project with limited workforce under extreme market pressure.

XPSF	XPDF	$\mathsf{NPV_C}$	$\mathsf{NPV_{XP}}$
1.4	15 %	474,817	341,932
1.8	15 %	474,817	477,233
1.8	25 %	441,177	477,233

The quantitative figures computed here using our economic model suggest the following *management guideline* for the sample project:

> If the size of the workforce does not allow to run the project with the maximum number of pairs, a manager should better add single programmers to maximize the degree of parallelism in the project instead of using XP.

This guideline holds in particular if the market pressure is only moderate. In such a setting, the potential speed and defect advantage of XP do *not* compensate the lack of parallelism in the XP project.

4.3 Sensitivity Analysis

The NPV ratio is sensitive to the XP speed factor and XP defect factor. For a fixed discount rate, we visualize the NPV ratio for different values of these parameters using a "checkboard." In each checkbord, we systematically vary the parameters over the following ranges:

Table 4: Varying XP speed and defect factor.

parameter	start .... stop	step
XPSpeedFactor	1.0 .... 2.0	0.1
XPDefectFactor	0 30	5

FIGURE 1 shows the checkboard when assuming a workforce of eight developers (four pairs) and a high annual DiscountRate of 50 percent. The numbers in the boxes specify the NPV ratios (in percentages). We have filled in only some of the numbers. The boxes are grey ▒ if the NPV ratio ranges between -10 and $+10$ percent, and white □ otherwise. In the upper half of the checkboard, the net present value of the conventional project is more than 10 percent larger than the net present value of the XP project; in the lower right corner of the checkboard, the XP project has a more than 10 percent higher value.

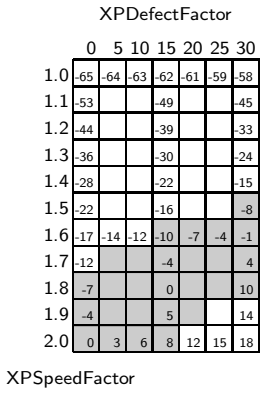

Figure 1: NPV ratios with four pairs under strong market pressure.

FIGURE 1 leads to a number of observations about how the value of the XP and conventional project (and hence, the NPV ratio) depend on the speed and defect factor:

- For a fixed defect factor, the NPV ratio increases as the speed factor increases; see for example the NPV ratios in the column XPDF = 15 percent.
- For a fixed speed factor, the NPV ratio increases as the defect factor increases; see for example the NPV ratios in the row XPSF = 1.6.
- Conventional development will outperform XP if the programmer pairs do not show a reduced defect rate; see the NPV ratios in the column XPDF = 0.
- XP will outperform the conventional project if the pairs work twice as fast as single programmers; see the NPV ratios in the row XPSF = 2.0.

The dependence of the NPV ratio on the XP speed and defect factor can be explained by going back to the formulas for the development time of the conventional and XP project, respectively. As the speed factor increases, the development time of the XP project decreases and hence its value increases; the value of the conventional project does not depend on the speed factor. On the other hand, as the defect factor increases the quality assurance time of the conventional project increases and hence its value decreases; the value of the XP project does not depend on the defect factor.

4.4 Developer Pool

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 (TaskLimit = 8) for the sample project, the manager has two options:

- He could run the project with up to eight single developers using a conventional process.
- He could run the project with up to sixteen developers who work in pairs using XP.

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

To make a decision whether to use XP or not, we apply our economic model assuming an annual DiscountRate of 75 percent, which corresponds to extreme market pressure. We then compare the conventional project with the maximum workforce of eight single developers against XP with the maximum workforce of eight *pairs*. TABLE 5 shows the results for different values of the XP speed factor XPSF and the XP defect factor XPDF.

Table 5: Net present value of sample project under extreme market pressure with maximum workforce.

XPSF	XPDF	NPV_C	NPV_{XP}
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

We conclude from TABLE 5 that XP outperforms the conventional project if the XP speed and defect advantage *both* are

significant (for example, XPSF = 1.4 and XPDF = 15 %). If the speed advantage is very large, XP outperforms the conventional project even when the defect advantage is small.

We visualize the dependence of the NPV ratio on the XP speed and defect factor using a checkboard, see FIGURE 2. For a fixed defect factor, the NPV ratio increases as the speed factor increases; a corresponding statement holds for a fixed speed factor. In addition, due to the stronger market pressure and larger number of pairs in the XP project, the area in the lower half of the checkboard where XP has a clear advantage over the conventional project has grown larger.

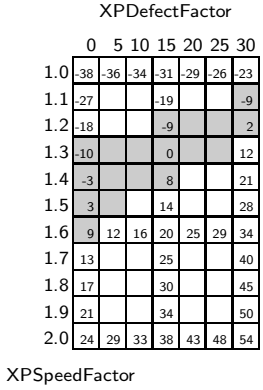

Figure 2: NPV ratios with eight pairs under extreme market pressure.

The quantitative figures computed here using our economic model suggest another *management guideline* for the sample project:

> A manager should consider applying XP if the market pressure is strong, his programmers are much faster when working in pairs as compared to working alone, and there is a sufficiently large workforce available to run the project with the maximum number of pairs.

For high discount rates, the speed and defect advantage of XP pairs come into full play. As a result, it can make sense to add developers to a project to form XP pairs because the increased personnel cost is more than balanced by the gain in market share.

4.5 Market Pressure

The market pressure plays an important role in the economic comparison of XP and conventional development. In general, the stronger the market pressure, the smaller the net present value of the project, no matter whether XP is used or a conventional process. This effect is apparent when comparing the numbers in TABLE 2, which correspond to a DiscountRate of 25 percent, with the numbers in TABLE 3, which correspond to a DiscountRate of 75 percent.

On the other hand, some computations have also shown that XP can pay off if the market pressure is strong. To study the impact of the market pressure on the NPV *ratio* more systematically, FIGURE 3 shows the checkboards for different project settings. Horizontally, the DiscountRate assumes the values 25, 50, and 75 percent. Vertically, the number of pairs in the XP project equals 4, 6, and 8. The number of single developers in the conventional project is always fixed at eight, assuming that one can't split the project tasks beyond the TaskLimit.

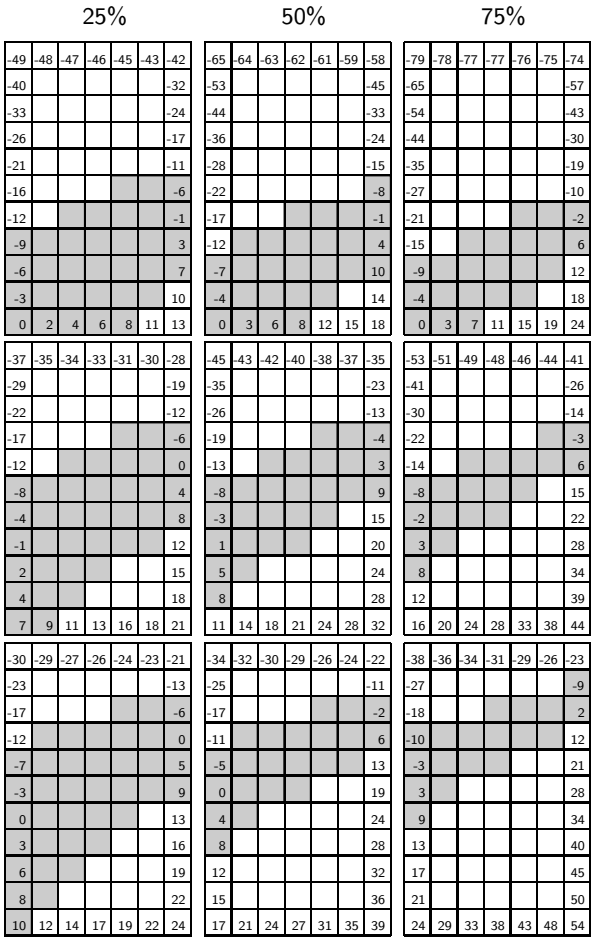

Figure 3: NPV ratios with varying discount rate and four to eight pairs.

The numbers in the upper left and lower right corner of each checkboard specify the maximum disadvantage, respectively, maximum advantage of XP over the conventional process. For a fixed workforce level, the spread between the max advantage and the max disadvantage of XP increases as the discount rate increases. In addition, the group of project settings where XP has a more than 10 percent advantage over conventional development grows as the discount rate increases, see the size of the white area in the lower half of the checkboards.

Suppose that the market pressure is strong (DiscountRate equals 50 percent). TABLE 6 shows in detail what happens to the NPV ratio if extra pairs are added to the XP project. In TABLE 6, the XP speed factor equals 1.6 and the XP defect factor equals 15 percent.

Table 6: Net present value of sample project under strong market pressure for varying number of XP pairs.

XP pairs	NPV_C	NPV_{XP}
4	540,578	488,645
6	540,578	569,556
8	540,578	612,241

For example, the NPV ratios are much better with six XP pairs than with four XP pairs. Adding XP pairs leads to a shorter time to market; under strong market pressure, a short time to market is decisive for the net present value of the project.

This result is also supported by FIGURE 3. With six or eight pairs, XP can outperform the conventional project for a smaller speed and defect advantage than with four pairs if the market pressure is strong. Recall that if the market pressure is only moderate, XP can outperform the conventional process only for a very high speed and defect advantage.

These findings suggest yet another *management guideline* for the sample project:

> The stronger the market pressure, the smaller are the number of pairs, the speed advantage, and the defect advantage which are required for XP to break even with the conventional process.

This guideline pins down the fact that the economic evaluation of XP depends not only on the XP speed and defect factor, but to a large extent on the *economic context* in which the project is carried out.

4.6 Incremental Delivery

Our economic project model assumes that the roll-out of the software is done at the end of the project and that payment is due at this point in time. Delivering the software at the end of the project is common practice in conventional software development. However, XP breaks with this practice as well. XP aims at delivering parts of the software, the so called *small releases,* from early on in the project cycle. In this subsection, we study the impact of *incremental delivery* on *both* conventional and XP projects.

With incremental delivery, the software is subdivided into "chunks" which are delivered as soon as they have been developed. For each chunk, the customer pays a fraction of the project's asset value. We assume that the payment is proportionate to the size of the chunk. This way, total payment for the project is split into a stream of small payments which is described by a sequence

$$\mathsf{PF} \;=\; (\,\mathsf{pf}_1,\;\ldots\;\mathsf{pf}_m\,)$$

of "product fractions" pf_k. Each fraction describes the size of the release relative to the whole product. For example, a project which delivers two equal-sized releases is described by the sequence $\mathsf{PF} = (\,0.5,\,0.5\,)$. The first release is delivered half time through the project, the second release at the end of the project. A one-release project is described by $\mathsf{PF} = (\,1.0\,)$.

To compute the net present value of a project with incremental delivery, payment k has to be discounted up to the point in time

$$\mathsf{DevTime}\,[\,\mathsf{k}\,] \;=\; \sum_{j=1}^{k} \mathsf{pf}_j \times \mathsf{DevTime}$$

when release k is delivered. We assume that the development cost for a small release is proportionate to the size of the release:

$$\mathsf{DevCost}\,[\,\mathsf{k}\,] \;=\; \mathsf{pf}_k \times \mathsf{DevCost}.$$

Finally, the net present value of a project which delivers in small releases is described by

$$\mathsf{NPV}\,[\,\mathsf{PF}\,] \;=\; \sum_{k=1}^{m} \Big\{ \frac{\mathsf{AssetValue} \times \mathsf{pf}_k}{(\,1 \;+\; \mathsf{DiscountRate}\,)^{\;\mathsf{DevTime}\,[\,k\,]}} \;-\; \mathsf{DevCost}\,[\,\mathsf{k}\,] \Big\}$$

TABLE 7 compares the two-release scenario $\mathsf{PF} = (\,0.5,\,0.5\,)$ with the one-release scenario $\mathsf{PF} = (\,1.0\,)$ for the sample project. The model parameters are chosen in such a way that the conventional project outperforms the XP project ($\mathsf{DiscountRate} = 50\,\%$, $\mathsf{XPSF} = 1.5$, $\mathsf{XPDF} = 15\,\%$, and $\mathsf{NumOfPairs} = 4$). For the two-release scenario, the columns in the table show the NPV of the first and second release as well as the NPV of the total project. The last column shows the NPV of the one-release scenario.

Table 7: Net present value of sample project with limited workforce and incremental delivery.

	two releases			one release
	1. release	2. release	total	
NPV_C	318,072	270,289	588,361	540,578
NPV_{XP}	283,457	228,238	511,695	456,476
ratio			-13 %	-16 %

Two effects can be observed:

- The net present value of the two-release project is higher than the value of the one-release project, regardless of which development process is being used.
- The relative advantage of the conventional project over XP *decreases* when delivering in two releases.

Both effects also occur when XP has an advantage over conventional development, see TABLE 8. The model parameters are $\mathsf{DiscountRate} = 75\,\%$, $\mathsf{XPSF} = 1.6$, $\mathsf{XPDF} = 15\,\%$, and $\mathsf{NumOfPairs} = 8$.

In the two-release scenario, one half of the asset value is returned after a *shorter* discount period as compared to the

Table 8: Net present value of sample project with maximum workforce and incremental delivery.

	two releases			one release
	1. release	2. release	total	
NPV_C	299,266	237,408	536,674	474,817
NPV_{XP}	323,762	285,404	609,166	570,807
ratio			14 %	20 %

one-release scenario. This higher cash inflow increases the business value of the project, regardless of the process used.

By the same argument, the process with the smaller project value in the one-release scenario has more to gain in the two-release scenario. For example, referring to TABLE 7, the additional return for the XP project in the two-release scenario (55,219 dollars) is larger than the additional return of the conventional project (47,783 dollars). Thus, the absolute value of the NPV ratio is smaller in the two-release scenario than in the one-release scenario.

FIGURE 4 shows the NPV ratios when using incremental delivery $\mathsf{PF} = (0.5, 0.5)$ for both XP and conventional development. The upper three checkboards correspond to a

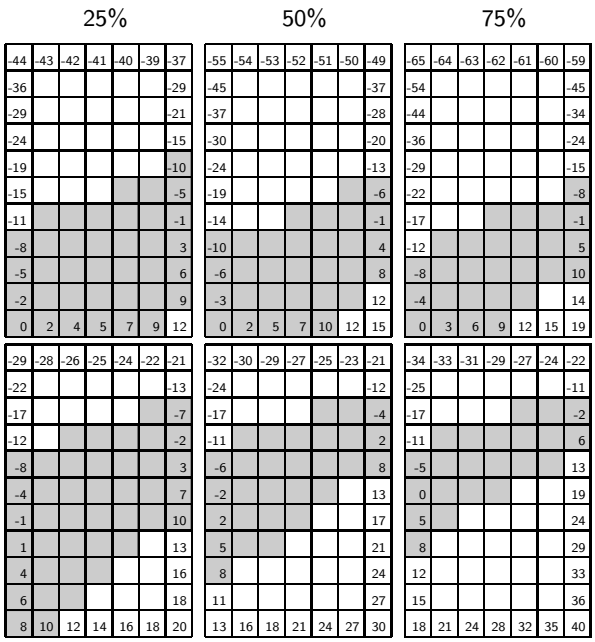

Figure 4: NPV ratios for incremental delivery.

limited workforce (eight single developers versus four XP pairs), the lower three checkboards correspond to a maximum workforce. As the checkboards show the NPV ratios, only the second effect can be observed: the relative advantage of the better process is smaller when delivering in two releases. Compared to the one-release scenario depicted in FIGURE 3, the grey zones have broadened and (up to rounding errors) the absolute NPV ratios are smaller.

The above discussion leads to the following *management guideline*:

> A manager should consider applying XP if that supports splitting the project into small releases better than a conventional process.

The stronger the market pressure, the more important it is to deliver and receive payment by the customer early. XP claims to be more suitable for delivering in small releases than conventional development. Currently, we lack empirical evidence collected from industrial projects of a realistic size in support of this claim; see [5] for an industrial XP experience report.

5. CONCLUSIONS

In this paper, we have presented an evaluation of XP from a project economics point of view. We have developed a novel economic project model which allows to study the impact of the key XP practices – Pair Programming and Test-Driven Development – on the business value of a project. Being based on the concept of net present value, our model also allows to consider the impact of the market pressure on the business value of the project.

Due to Pair Programming, personnel cost is doubled in a XP project as compared to a conventional project. Also, the effort spent on continuously adapting and running the test cases in Test-Driven Development is likely to add to the task completion times in a project. On the other hand, pairs have a speed advantage over single developers, and both Pair Programming and Test-Driven Development increase the code quality. Therefore, the increased cost of applying XP can be balanced – under suitable economic conditions – by its faster time to market and reduced effort for late quality assurance.

We have applied our economic model to a hypothetical, yet realistic sample project. More precisely, we have compared the net present value of the sample project when using a conventional process against the net present value of the sample project when applying XP. We have studied different project scenarios and provided a detailed sensitivity analysis with respect to the model parameters. The XPSpeedFactor, XPDefectFactor, and DiscountRate each have a strong impact on the business value of the XP project. For some project settings, XP outperforms the conventional process, but there also are settings where the opposite is true.

The results of our computations provide clear management guidelines when to use XP – or better not:

- If the size of the workforce does not allow to run the project with the maximum number of pairs, a manager should better add single programmers to maximize the degree of parallelism in the project instead of using XP.
- A manager should consider applying XP if the market pressure is strong, his programmers are much faster when working in pairs as compared to working alone, and there is a sufficiently large workforce available to run the project with the maximum number of pairs.

- The stronger the market pressure, the smaller the speed advantage, the defect advantage, and the number of pairs which are required for XP to break even with the conventional process.

- A manager should consider applying XP if that supports splitting the project into small releases better than a conventional process.

Other projects will differ from the sample project in the values of some model parameters, such as the ProductSize, the AssetValue, or the Productivity. Since there is nothing special about the sample project and its size, we expect these guidelines to be valid for other projects as well, although the particular numbers will change, of course.

Currently, our model does not take into account some management issues which typically occur when the project exceeds a certain size, such as high staff turnover, increased communication overhead, and overtime work. Such issues play a role in both conventional and XP projects. We also do not model the tradeoffs involved in applying *refactoring,* which is another important XP technique.

Another topic which might be interesting to study from an economics viewpoint is customer satisfaction with the functionality of the software. As opposed to conventional projects, XP compiles a stack of *story cards* each of which captures one particular requirement in the form of a usage scenario. The story cards are written together with the customer, who is supposed to be on the project all the time. XP does *not* maintain a requirements specification document, nor a high-level design document. This fact might make it difficult to guarantee that the software as a whole actually fulfils the customer's needs [5].

We think that it is important to understand that an objective evaluation of XP depends not only on certain process metrics, such as the XP speed and defect factor, but to a large extent on the *economic context* in which the project is carried out. We consider concepts from economics (such as net present value) to be just the right vehicle to combine software engineering metrics in order to assess the tradeoffs involved in a new development paradigm. Our computations show that the interplay between the different metrics for XP and the economic project context is rich.

The business value of XP as compared to conventional development much depends on how large the speed and defect advantage of XP pairs actually are. Currently, our knowledge about the true values for these parameters is severly limited.

For one thing, Pair Programming and Test-Driven Development have only been studied *separately* so far. Recall that Pair Programming and Test-Driven Development have an opposite impact on the task completion times, but a similar impact on the code quality. Hence, it remains unclear how to combine data about the PairSpeedAdvantage, PairDefectAdvantage, and TestDrivenSpeedFactor in order to quantify the XPSpeedFactor and XPDefectFactor. In addition, quantitative data about Pair Programming and Test-Driven Development is scarce to date.

The XPSpeedFactor and XPDefectFactor can be determined empirically in the same way as we study Pair Programming or Test-Driven Development. We consider it an important task to collect reliable empirical data about how large the speed and defect advantage of XP pairs actually are. We also must study how well XP supports delivering small releases. Most notably, we lack empirical data from industrial XP projects having a realistic size. From this point of view, our paper gives a clear direction for future empirical research about XP.

6. 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] A. Elssamadisy and G. Schalliol. Recognizing and responding to bad smells in extreme programming. In *International Conference on Software Engineering ICSE-24*, pages 617–622, Orlando, Florida, USA, May 2002.

[6] H. Erdogmus. Comparative evaluation of software development strategies based on net present value. In *International Workshop on Economics-Driven Software Engineering Research EDSER-1*, Los Angeles, USA, May 1999.

[7] 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-1*, Los Angeles, USA, May 1999.

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

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

[10] M. Müller and O. Hagner. Experiment about test-first programming. *IEE Proceedings on Software*, 149(5):131–136, Oct. 2002.

[11] 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-4*, Orlando, Florida, USA, May 2002.

[12] M. M. Müller and F. Padberg. About the return on investment of test-driven development. In *International Workshop on Economics-Driven Software Engineering Research EDSER-5*, Portland, Oregon, USA, May 2003.

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

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

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

[16] L. Williams and H. Erdogmus. On the economic feasibility of pair programming. In *International Workshop on Economics-Driven Software Engineering Research EDSER-4*, Orlando, Florida, USA, May 2002.

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