Analyzing the Cost and Benefit of Pair Programming Revisited

Lev Faynshteyn, Vojislav B. Mišić, Jelena Mišić
Ryerson University, Toronto, Canada

Contribution to the State of the Art
DOI: **10.7251/JIT1201014F** UDC: **502.171:551.782**

Abstract: Pair programming has received a lot of attention from both industry and academia, but most paper focus on its technical aspects, while its business value has received much less attention. In this paper, we focus on the business aspects of pair programming, by using a number of software development related met rics, such as pair speed advantage, module breakdown structure of the software and project value discount rate, and augmenting them by taking into account the cost of change after the initial product release and inherent non-linearity of the discount rate curves. The proposed model allows for a more realistic estimation of the final project value, and the results of System Dynamics simulations demonstrate some useful insights for software development management.

Keywords: Pair Programming, Extreme Programming (XP), System Dynamics, Waterfall, Cost of Change

Introduction

Pair Programming (PP) is one of the key paradigms in Extreme Programming (XP). It stipulates that any coding task should always be performed by a pair of programmers working at the same computer using only one set of input devices. This approach has been demonstrated [7][11][21][22] to provide tangible benefits in such areas as design and code quality (fewer bugs per line of code), problem solving (two heads are better than one) and general satisfaction with a job well done (people like to share responsibility, which in turn makes them feel more confident and comfortable). At the same time, benefits related to the improved productivity have not been fully corroborated [10], and experience has shown that to leverage the full potential of Pair Programming, that is to keep the original level of task parallelism in any given company while taking advantage of all the benefits, the number of developers has to be doubled, which in most cases also means doubling the personnel costs. Naturally the question arises when and under what conditions the additional expenses are justifiable. To answer this question Padberg and Müller created a mathematical model [18] which is based on three categories of metrics used as input to the model:

- *Process Metrics:* productivity of a single developer, pair speed advantage (PSA), defect density of code, pair defect advantage (PDA) and defect removal time.
- *Product Metrics:* product size and module breakdown structure of the software.
- *Project Context Metrics:* project value discount rate, initial asset value, number of single developers, number of programmer pairs, developer and project leader salaries, monthly working hours.

By analyzing these metrics and studying their relationships they came up with a mathematical expression for Net Present Value (NPV) of a software project, which in a nutshell repre sents the initial monetary value of a project (AssetValue) discounted at a certain rate (DiscountRate) minus the development expenses (DevCost) throughout the entire duration of the development process (DevTime):

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

By adjusting the model's input parameters (mostly PSA, PDA and MP) they collected results representing various configurations for both projects developed under traditional software development practices (e.g. a Waterfall process) and projects that utilized Pair Programming. Their conclusion was that for projects where PSA is moderate and MP is not very high, conventional development methods will produce better financial returns. In fact, even in cases where discount rate figures are very high (e.g. 75% per year), PSA is large (such as 1.8 times of a single developer productivity) and PDA is quite significant (15% or more less bugs in the code), Pair Programming would just break even with conventional practices. However, successful real life applications [5][11][12][20] of XP practices offer ample evidence that Pair Programming does work and is certainly economically feasible. Naturally a question arises whether the original model is missing on some aspects of XP in general and Pair Programming in particular that might change the balance in favor of the latter. The following sections will try to address this question.

Motivation

According to the original model, Pair Programming will be economically feasible only in extreme cases, where time to market is absolutely critical (i.e. project value discount rate is extraordinarily high). However this, as the examples in the previous section show, is often not true and XP, and Pair Programming as an inherent part of it, are used for projects of all scales and time durations, with many showing positive results in terms of both productivity and profitability. While looking for an explanation of this discrepancy we came to realize two things:

First of all, discount rate in the original paper is always a constant value. This seems unrealistic since discount rate itself is subject to many factors. For instance it would be reasonable to assume that for a brand new product not only would it be very high, but after some point it would accelerate at a much higher rate than initially due to the fact that market rivals would have released or would be drawing ever so closer to releasing a competing product. At the same time for a well established product the acceleration would be very slow at first since the established user base would be unwilling to upgrade too often and conversely they would be willing to wait for quite a long time for an update for a product that has already proved itself. However after a certain moment in time it would also start to accelerate at a faster rate, since going beyond a certain point in time without a new version would test customers' patience. These ideas are in fact confirmed by the real world data [14] and thus this change will be a good candidate for an improved model.

A second and probably more important observation was that the original model did not consider the cost of change (CoC) of the code after the initial release date. In 1981 Barry W. Boehm did a study [3] of the cost of change ratio between implementing a feature or fixing a bug in production vs. requirements stage and found it to grow exponentially with time. Even for projects of moderate size it could be very high (up to 100 times and more). His much more recent book [4] confirms these numbers. This makes sense for traditional approaches where the requirements and features are for the most part determined once at the beginning of a project and stay the same throughout the whole development cycle until the software is released. Any new feature requests are being deferred until after the release, thus making their implementation potentially very difficult and labor inten-sive.

On the other hand, in XP the development process starts with only a general idea about how the final product will look like or function, and is constantly refined by means of customer feedback. It thus allows in a way to defer the cost of making big and costly decisions early on and to have the best chance that once these big decisions are made they would be the right ones. This is the premise on which Kent Beck in [2] based his argument that for XP the curve of cost of change is way more shallow than for the traditional methods, and the actual costs of changes in production vs. requirements phase can be as low as five and would stay close to these low values for extended periods of time (in fact this might be the very reason why XP is economically feasible).

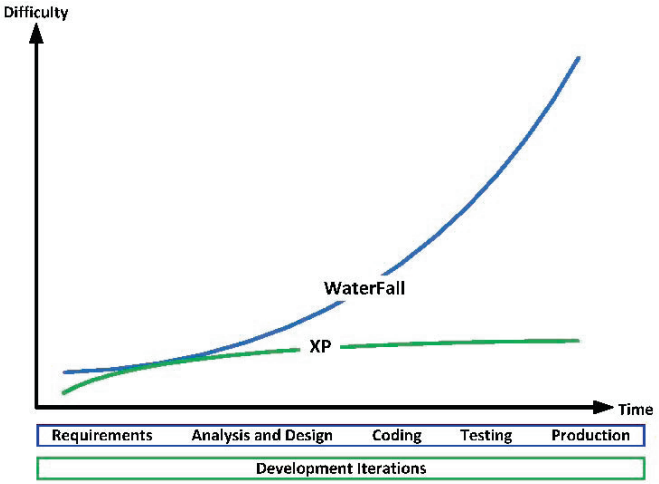

Figure 1. Cost of Change in cases of a Waterfall and an XP development approaches (after[2])

Fig. 1 shows a qualitative comparison of the curve of CoC for a hypothetical software project done using a traditional (e.g. Waterfall) and an agile (e.g. XP) approach.

An example in [4] of a project (TRW CCPDS-R) that employed an innovative hybrid approach to the development process, in which both the traditional and agile practices were employed, seems to confirm this supposition, as the design, implementation and maintenance changes throughout the lifecycle of the project remained at a very low level.

Considering all of the above it seems reasonable that a new version of the model should include a CoC metric since it seems quite probable that it might significantly affect the results of simulations.

To implement a modified version of the model a System Dynamics approach was chosen. System Dynamics is a proven technique that allows flexible and efficient exploration and analysis of the behavior of complex systems over time by describing them in terms of interconnected elements that continually interact with each other and the outside world to form a unified whole [15]. Due to the inherent dynamic nature of software development processes and their often complex interrelations, Systems Dynamics has been long recognized as a very potent approach to modeling of the former, often resulting in exposure of surprising non-linearities in models of even modest dimensions.

Related Work

To date, many areas of agile development practices have been analyzed in numerous publications, spanning all the way from studies in the area of psychology and pair compatibility [9] to more generic evaluation of the effects of the learning phase in the context of XP on productivity [17] to the attempts to model the entire XP development process [18]. However, since the scope of this paper mainly deals with the economic benefits of Pair Programming, we will focus our attention on the most relevant publications.

In [19] Padberg and Müller extended their model for NPV to include the effects of the learning phase (inherent to the Pair Programming) on the final value of the project. Their results showed that due to the fact that the learning phase typically incurs a onetime cost and the learning process itself does not take long, the overall effect of the former is typically minimal and amounts only to a few percent of the total project cost: the learning overhead did not exceed 10% even in cases with very high staff turnover. Thus, the estimates and conclusions in the original paper remained largely unchanged.

In [8] the authors propose a metrics-oriented evaluation model that allowed them to assess a chosen development model based on the project's predicted NPV value. The proposed NPV formula takes into account such variables as development time and cost, asset value, operation cost, flexibility value and product risk. However, this model deals with the high-level representation of the underlying development process and as such does not reflect the intricacies of any particular approach (whether it be traditional or agile method).

In [13] the attempt is made to test the validity of the supposition that the cost of change curve in case of the agile development practices is indeed much flatter than that of the traditional approaches. The authors employ a System Dynamics approach to build a fairly involved model in which the main criterion of effi ciency is the number of requested vs. the number of implemented user stories. However, the results and conclusion sections are very scarce and superficial and fail to elaborate on the actual out-

come of the simulation runs, thus leaving the question unanswered.

Various forums and software development websites (e.g. [1][6]) have discussions related to the nature of the cost of change and its effects on the development costs in particular and feasibility of different project development strategies as a whole. However, as of today, the opinions vary wildly and conclusions seem to be based on mostly anecdotal evidence and common sense. Thus, the authors of this paper understand that the proposed enchantments are merely an educated guess and reflect their subjective opinion on the matter.

The Extended System Dynamics Model

The implementation was done in GoldSim, a simulation package by GoldSim Technology Group, which is a quite powerful and feature rich Monte Carlo [16] simulation suite. To account for both traditional and agile approaches (later referred to Waterfall and XP respectively) two separate models were created. To keep the results of the simulations consistent both models share the same set of input metrics, which in their turn, to make results comparable to the ones in the original paper, were kept the same (see the original paper [18] for details and the rationale behind selecting the particular values) and are presented in Table 1.

Both models contain elements that correspond to a defect generation process (defects are produced at a defect density rate depending on the volume of the written code at any given moment). These bugs cause additional workload for the developers (their approximated number of LOC is added to the initial prod uct size) and thus, the defect removal time metric of the original model is implicitly expressed through a dynamic feedback loop.

In addition, both models implement a concept of a code backlog. The idea behind it is that as time goes by customers will be asking to introduce new features into the system that is currently being developed. In case of a Waterfall process all of these features will be delayed until after the initial release, thus creating a code backlog, which basically consists of a sum of all approximated numbers of LOC needed to implement all of the features at the time when they are requested. Depending on a chosen market pressure curve and a product release date, this aggregate number will be multiplied by the CoC value and the work will continue, marking a new development period with additional expenses for the company. In case of Pair Programming the backlog will be much smaller since user requirements will be, for the most part, implemented and integrated into the system during its development stage. Note that not all of the fea-

Table 1. Input metrics and their values (after [18]).

Input Metric Name	Value/Range	Comments
Productivity of a Single Developer	350	Lines of code (LOC) per month
Pair Speed Advantage (PSA)	1.4-1.8	A speed advantage of a pair of programmers as compared to a single programmer
Product Size	16800-100000	Initially estimated size of the project in LOC
Defect Density	0.03	Defects per LOC
Pair Defect Advantage (PDA)	5% - 15%	A quality advantage a pair of programmers has as compared to a single programmer
Module Breakdown Value	8	Maximum developer/pair parallelism in the scope of the project
Discount Rate	0 - 1	Various linear and non-linear values as functions of time (see below for details)
Asset Value	1000000 – 3000000	Initial amount agreed upon for the development of a project in $USD
Number of Single Developers	Equal or less than a Module Breakdown Value	
Number of Programmer Pairs	Equal or less than a Module Breakdown Value	
Developer Salary	50000	in $USD
Project Leader Salary	60000	in $USD
Monthly Working Hours	135	

Table 2. Newly introduced input metrics and their values approaches.

Input Metric Name	Value/Range	Comments
User Stories	Random	A new user story is randomly generated (uniform distribution) approximately once every 1.5 month
Feature Size	Random	A size of a new feature in LOC is generated randomly (exponential distribution) with big features progressively less likely to occur vs. small features
Feature to Backlog Ratio	0.2 – 0.3 for Waterfall 0.05 for XP (PP)	Represents percentage of the Code Backlog that will actually be implemented after the initial release
Cost of Change Curve	X^3 growth for Waterfall ln(X) growth for XP (PP)	Changes with time according to the relevant equation. Please see the models for the actual values.

tures in the code backlog will be implemented (some of them will be covered by other features, others will be dropped, etc.). This fact is represented by another input metric called Feature to Backlog Ratio. Table 2 lists the newly introduced metrics that are used in the CoC related part of the model.

One more important difference between the new and the original model lies in the fact that for the new Pair Programming model a concept of refactoring has also been implemented. Refactoring happens whenever a new bug is reported, a new feature is added or when the number of both bugs and new features introduced into the system exceeds a certain value [2] (this in XP circles is sometimes referred to as "when the code start to smell"; in this particular case the values are 5 for bugs and 10 for features).

The basic idea behind both models is the same: the initial estimated size of the project gets chipped away at a development rate that depends on the productivity and team size. As the code is being generated, bugs start to appear according to the predefined defect density and features are requested according to the predefined random distribution. The weights of the bugs and features (that is how many LOC each of them will take to fix/implement) are also randomly determined according to separate random distributions. The resulting values are added to the total pull of work (for XP features, for the most part, are added right away, for Waterfall they go into the code backlog). When the size of the project goes down to zero (that is there is no more work to be done) for the first time, we reach a stage of the first release. At this point, a second part of the model activates that determines how many LOC it would take to clear up the code backlog considering the current value of the CoC. The project size depository gets refilled with the newly calculated value for the LOC and the work resumes in the same way as earlier, except that new features are no longer accepted. Note that it is also possible to run multiple realizations for each of the models by specifying the number of Monte Carlo stages. This allows us to see how such random input variables as features and bugs affect the results of the simulations.

Results

Since the newly created Waterfall model is basically identical to the model in the original paper [18] we can use it as a gauging device to see if the results produced by it are comparable to the reference results in the original publication. Using the original model values (see Table 1.) for a product with 16800 LOC, an asset value of 1000000 dollars and

Table 3. Results of the simulations

PSA	PDA	System Size	Asset Value	Discount Rate	NPV_{WFR} in Days	NPV_{WFB} in Days	NPV_{PPR} in Days	NPV_{PPB} in Days
1.4	5%	16800	1000000	Constant 10% per year	**710016** in **192**	**520663** in **318**	**355379** in **247**	**292323** in **271**
1.4	15%	16800	1000000	Constant 10% per year	**710016** in **192**	**520663** in **318**	**360379** in **245**	**297820** in **268**
1.8	15%	16800	1000000	Constant 10% per year	**710016** in **192**	**520663** in **318**	**704761** in **113**	**704085** in **114**
1.4	5%	30000	3000000	Constant 10% per year	**2085645** in **455**	**1186641** in **903**	**1650857** in **435**	**1566864** in **462**
1.8	15%	30000	3000000	Constant 10% per year	**2085645** in **455**	**1186641** in **903**	**2050841** in **306**	**2013761** in **318**
1.4	15%	30000	3000000	**New** product (very steep drop after about 200 days)	**369769** in **455**	**-834246** in **903**	**48425** in **428**	**- 120590** in **454**
1.4	15%	30000	3000000	**Mature** product (slow acceleration of pressure)	**2254082** in **455**	**1279895** in **903**	**1833987** in **428**	**1759799** in **454**

a constant discount rate of 10% per year, the reference NPV for conventional development process was estimated at 723,463 dollars. The modified Waterfall model produced an average NPV value (for the release date) of 710016 dollars, which considering the random nature of the simulation is close enough. Having established that reference point, let us now see how different discount rate, project size, PSA and PDA values affect the results of the simulations.

To compare the results of the new model simulations to the results presented in the original paper [18] a number of runs (each consisting of 100 realizations) with different input values were executed. The results are given in Tables 3 (all of the parameters, except the ones listed in the table, were kept the same throughout all of the runs), where NPVWFR, NPVWFB, NPVPPR, NPVPPB are NPV values for Waterfall Release, Waterfall Backlog, Pair Programming Release and Pair Programming Backlog milestones respectively.

The first batch of experiments was run at a constant yearly discount rate of 10%. As can be seen from the table, for a project of a relatively small size of 16800 LOC, conventional development methods prove superior when PSA is kept at a reasonable level of 1.4 (PDA variations have very limited effect on the results, thus they are largely disregarded in the discussion). So far this is in line with the results of the original paper, though one interesting point to note is that even at this low level of PSA, the Pair Programming model has finished processing its code backlog considerably earlier than the Waterfall model. Increasing PSA to 1.8 (the same highest value that was used in the original paper) changes the picture quite a bit: now Pair Programming basically breaks even with Waterfall model for the release date in terms of money, and considerably outperforms it in terms of simulation durations in both release and backlog cases.

Increasing the project's size by roughly three times and performing the same tests shows us that for bigger projects (i.e. those that will take longer to deliver) even with modest levels of PSA, Pair Programming often breaks even (a little less money at the release date, but on the other hand release is done some-

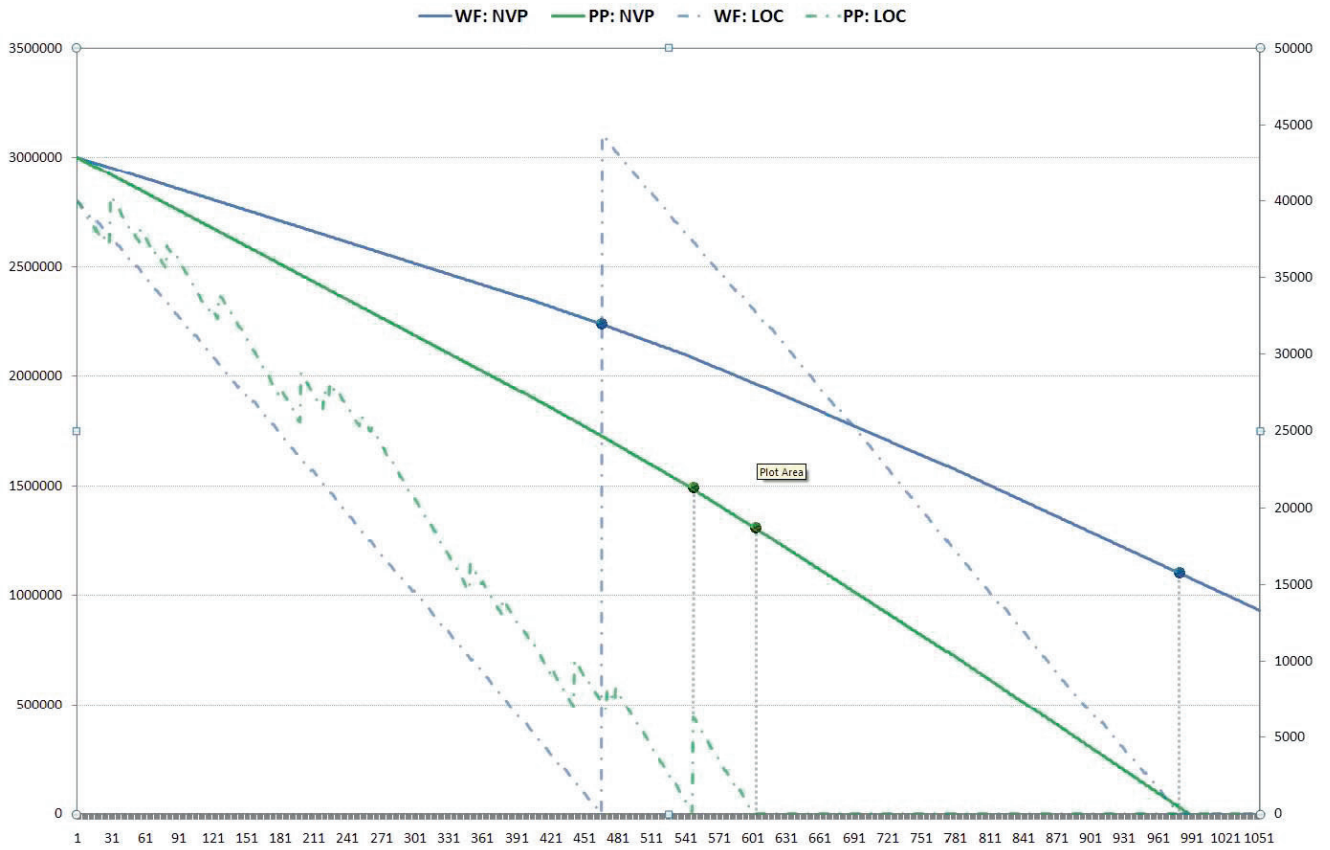

Figure 2. Examples of NPV values obtained using the Pair Programming (green) and Waterfall (blue) models

what earlier) or actually pulls ahead of a conventional model. Due to the ever accelerating CoC curve for the Waterfall model, this gap will be only growing. For example increasing the size of a project to 50000 LOC brings us to a situation where a team of 8 programmers is unable to clean up the Code Backlog within a maximum simulation period of 1050 days, whereas a Pair Programming team can finish it in less than 600 days with 1.2 million dollars in profits.

The simulations with non-linear discount rates confirm earlier observations. Namely, Waterfall might win some runs on the release day, but it will lose everything later due to a huge code backlog. Note that the Feature to Backlog ratio was actually kept at a low level of 0.2 (only 20% of the feature related code was implemented in the course of the backlog stage), thus actually favoring the Waterfall model.

Fig. 2 shows examples of two realizations from one of the runs in case of a mature product for both the Pair Programming and Waterfall models. The solid green and blue lines represent the NPV values expressed in $USD for Pair Programming and Waterfall model, respectively. The first solid dot on each line corresponds to the NPV value at the moment of the initial software release, and the second solid dot corresponds to the NPV value at the moment when the entire code backlog has been taken care of. Similarly the dash-and-dot green and blue lines represent the amount of coding that still remains to be done at any given moment in time expressed in LOC. The left vertical axis shows the amount of $USD, the right vertical axis is the LOC number, and the horizontal axis is the time of the simulation expressed in days.

Conclusions and Future Work

As the results of the previous section show, conventional development approaches such a Waterfall model can prove to be a better choice in cases of a smaller project with relatively low rate of new feature requests. However even with a low feature request rate used in the models (1 new feature every 1.5 months) and a low Feature to Backlog ratio value of 0.2, it is struggling to keep up with the Pair Programming model. Even for modest values of PSA (1.4 is actually a very realistic value [2][18] confirmed by several sources) Pair Programming proves to be a better approach: the initial release dates are close enough to the ones obtained using conventional methods, while the ability to quickly clean up the backlog will be a real boon for any company. Also note that even if Pair Programming losses on paper moneywise, it often delivers the product earlier (for example in Table 3 there are cases where the NPVPPR is less than NPVWFR, but "time to market" is shorter) and though it is not quantifiable in the scope of this model it has to be worth something in real life.

That having been said, the results of this simulation should be taken with a grain of salt. First, many values, especially those related to the new feature generation and code backlogging processes, are no more than educated guesses, which are mostly based on the authors' industrial experience. In real life, they are likely to vary considerably from project to project and company to company. However, the results are representative of qualitative trends.

Second, there is only limited evidence of what the actual CoC curves look like. In real life, too many variables, such as coding and managerial practices, technology and tools used, programmers' compatibility and expertise, etc. can affect their actual shape and values and definitely more research based on real life data is needed in this field.

Our future research will look into improvements of the model, for which there are quite a few possibilities. For instance, a feature generation rate can be made a function of the discount rate, thus reflecting the fact that customers usually want to see in the developed software the same or similar features to the ones competitors already have in theirs. At the same time this rate will have to be checked against some kind of a deadline/cutoff condition. Otherwise we might end up being swamped with features without hope of ever finishing the project. Finally, different parts of the model can also be broken down into smaller pieces to reflect the underlying processes with greater detail and accuracy. For instance, such an aspect of Pair Programming as pair switching and associated learning curve can be included in the model.

This task is simplified by the fact that the process of System Dynamics model conversion between different modeling suites is a pretty straightforward one, and thus any aspect of XP software development cycle implemented as a System Dynamics model to date can be readily converted into the necessary format and integrated with the current model with only minor investments in terms of both time and efforts.

References

[1] Ambler, S. W. (2003). Examining the Agile Cost of Change Curve, http://www.agilemodeling.com/essays/costOfChange.htm

[2] Beck, K. (1999). Extreme Programming Explained:Embrace Change. Addison Wesley.

[3] Boehm, B. (1981). Software Engineering Economics.Prentice-Hall.

[4] Boehm, B. and Turner, R. (2003). Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Professional.

[5] Capiluppi, A., Fernandez-Ramil, J., Higman, J., Sharp, H. C. and Smith, N. (2007). An Empirical Study of the Evolution of an Agile-Developed Software System. 29th Int. Conf. on Software Engineering ICSE 2007, Minneapolis, MN.

[6] Cockburn, A. (2000). http://xprogramming.com/articles/cost_of_change/

[7] Cockburn, A. and Williams, L. (2000). The costs and benefits of pair programming. eXtreme Programming and Flexible Processes in Software Engineering XP 2000 .

[8] Erdogmus, H. (1999). Comparative evaluation of software development strategies based on Net Present Value. ICSE Workshop on Economics-Driven Software Engineering, Los Angeles, CA

[9] Hannay, J. E., Arisholm, E., Engvik, H. and Sjøberg, D. I. K. (2010). Effects of personality on pair programming. IEEE Transactions on Software Engineering 36(1):61-80.

[10] Hannay, J. E., Dybåa, T., Arisholm, E. and Sjøberg, D. I. K. (2009). The effectiveness of pair programming: a metaanalysis. Information and Software Technology 51(7):1110-1122.

[11] Hulkko, H., & Abrahamsson, P. (2005, 15-21 May 2005). A multiple case study on the impact of pair programming on product quality. Int. Conf. on Software Engineering ICSE'2005, St. Louis, MO.

[12] Khalaf, S. J. and Maria, K. A. (2009). An Empirical Study of XP: The Case of Jordan. Int. Conf. Information and Multimedia Technology ICIMT'09, Jeju Island, Korea.

[13] Kuppuswami, S., Vivekanandan, K. and Rodrigues, P. (2003). A system dynamics simulation model to find the effects of XP on cost of change curve. 4th Int. Conf. Extreme Programming and Agile Processes in Software Engineering XP 2003, Genova, Italy.

[14] Little,T. (2004). Value creation and capture: a model of the software development process. IEEE Software 21(3):48-53.

[15] Madachy, R. J. (2008). Software Process Dynamics. Wiley.

[16] Metropolis, N. and Ulam, S. (1949). The Monte Carlo Method. J. of the American Statistical Association 44(247):335–341.

[17] Mišić, V. B., Gevaert, H. and Rennie, M. (2004). Extreme Dynamics: Towards a System Dynamics Model of the Extreme Programming Software Development Process. ProSim'04 Workshop, Edingburgh, UK.

[18] Padberg, F. and Müller, M. M. (2003). Analyzing the cost and benefit of pair programming. IEEE METRICS pp. 166-179, Sydney, Australia.

[19] Padberg, F. and Müller, M. M. (2004). Modeling the impact of a learning phase on the business value of a pair programming project. 11th Asia-Pacific SoftwareEngineering Conf., Busan, Korea.

[20] [20] Shaochun, X. and Rajlich, V. (2006). Empirical Validation of Test-Driven Pair Programming in Game Development. IEEE/ACIS ICIS-COMSAR 2006, Honolulu, HI.

[21] Williams, L. (2000). The Collaborative Software Process. PhD dissertation, University of Utah, Salt Lake City, UT.

[22] Williams, L., Kessler, R. R., Cunningham, W. and Jeffries, R. (2000). Strengthening the case for pair rogramming. IEEE Software 17(4), 19-25.

[23] Yang, Y. and Bosheng, Z. (2009). Evaluating Extreme Programming Effect through System Dynamics Modeling. Int. Conf. on Computational Intelligence and Software Engineering (CiSE 2009), Wuhan, China.

Submitted: April 23, 2012
Accepted: June 7, 2012