Design of an Experiment for Quantitative Assessment of Pair Programming Practices

Giancarlo Succi **Milorad Stefanovic**
Dept. of Electrical and Computer Engineering
University of Alberta
Edmonton, AB, Canada
+(1-780) 492-7228
{giancarlo.succi, misha}@ee.ualberta.ca

Michael Smith **Richard Huntrods**
Dept. of Electrical and Computer Engineering
University of Calgary
Calgary, AB, Canada
+(1-403) 220-6142
{smith, huntrods}@enel.ucalgary.ca

ABSTRACT
Anecdotal evidence demonstrates success of extreme programming practices in a portion of the software industry. It has also been argued that pair programming, as a part of the extreme programming process, yields higher quality software products in less time. On the other hand, these principles are sometimes questioned with respect to resource allocation and management issues.

Although precise information about benefits and costs of the extreme programming practice represents a critical guideline for improvement of software quality, there has been little work on the subject beyond subjective reports and a study in an academic environment.

In our work, we propose an experimental framework to quantify benefits and costs of the pair programming practice and compare design aspects of the resulting software products and their defect behavior. For this purpose, we use a set of object-oriented metrics and software reliability growth models based on occurrence of service requests.

Keywords
eXtreme programming, software engineering metrics and models, experimental design

1 INTRODUCTION

Any software development process has to deal with change in order to satisfy rapidly changing requirements and technologies. Virtually all companies in today's highly competitive industry are putting a lot of effort and resources in meeting and exceeding customer expectations. Lightweight software development methodologies, such as eXtreme Programming (XP), offer a promising way for successful dealing with changes.

There is anecdotal evidence of success of the extreme programming practice in a portion of the software industry [10]. An important concept in XP methodology is Pair Programming (PP), which defines software development as an activity practiced by two developers working together at one machine [4].

It has also been argued that pair programming, as a part of the extreme programming process, yields higher quality software products in less time. In addition, some reports suggest higher developer satisfaction and confidence as a result of this software development process.

On the other hand, these principles are sometimes questioned with respect to resource allocation and management issues [4,10].

Just like any other software development methodology, XP and PP are not equally appropriate for every environment. Although precise information about benefits and costs of the extreme programming practice represents a critical guideline for improvement of software quality, reduction of development costs, and improvement of both developers' and customers' satisfaction, there is a little work on the subject beyond subjective reports and a study in an academic environment [9].

Williams *et al.* [24] use combination of an experiment conducted at academic environment and the anonymous questionnaire to support claims of PP about higher quality of produced software, faster (at lower price), and with higher programmers' confidence. Details about this experiment ran at the University of Utah are described in [25].

The challenge is to determine which methodology should, and can be successfully applied to a specific environment to achieve best results, *i.e.*, an increased quality at the same or lower costs. To answer some of these questions objectively, we propose a framework for an experiment in an industrial environment, introducing the pair programming practice to the existing software process.

The software metrics and models are invaluable for software process characterization and improvement. Our goal is to set up the environment and select a well-defined set of product and process metrics to support an objective assessment of the costs and benefits of the PP practices and other possible changes to the software development process in a company.

In order to quantify benefits and costs of the pair programming practice, we compare design aspects of the resulting software products and their defect behavior. For this purpose, we use a set of object-oriented metrics to measure design attributes such as coupling, complexity, and size. To accurately describe the differences between the two processes, particularly with respect to occurrence and types of service requests (SRs) and effort necessary for fixing them, we use software reliability growth models (SRGM) [22].

The rest of the paper is organized as follows: Section 2 represents an overview of the background issues relevant for this study, such as object-oriented metrics and models and software reliability growth models. Section 3 proposes a framework for experimentation. Section 4 provides more details on data collection process. Finally, Section 5 concludes this study and provides suggestions for future work

2 BACKGROUND

XP is a lightweight software development methodology. It is designed for relatively small teams of up to 10 people [4].

XP is built on few basic principles: simple design, small releases, continuous restructuring and integration, aggressive testing, pair programming and collective code ownership. Proponents of XP insist on the need for its full adoption to make it work. However, in order to introduce these principles to an existing software development process and maintain the control over the process, it is necessary to adopt these principles in a controlled way, probably one at a time [4].

Significant part of the XP success could be attributed to the PP practice. Not only that working in pairs has great potential to improve the communication within the team, but it also provides a way for continuous review of both design and code. Having in mind that cost of correcting a defect grows exponentially with time when it is detected, an effective review process clearly helps in reducing development and maintenance costs. Combined with the collective ownership component, PP also creates a positive pressure, helping achieve better software quality.

Another important aspect of PP is the dynamic change of pairs. This practice leads to the efficient exchange of knowledge and experience between the team members. In this way, the risk of turnover is also reduced, since the knowledge about the system is distributed among the members of the team.

To objectively determine the effects of pair programming practice effects in an industrial environment, we propose a framework for an experiment based on object-oriented metrics and models and the occurrence of software service requests.

Issues in Object Oriented Metrics and Models

In this study we use the relatively simple and well-understood CK metrics suite proposed by Chidamber and Kemerer [5]. This set of six metrics shows a good potential as a complete measurement framework in an object-oriented environment [16].

Based on the set of design metrics, appropriate software engineering models can be built to link internal design aspects of the software product with its defect behavior. Number of defects for a class represents such external count metrics measured on an absolute scale [12].

Although there is no doubt that some analyses could provide useful results even though all theoretical assumptions are not met in reality [5], it is very important to apply statistical method with assumptions closest to the empirical system. This is especially important having in mind that inappropriate methods, such as the common practice of treating count variables as continuous [11,14], may result in inefficient and biased models and wrong or misleading results. Discreteness of the dependent variable also leads to conservative confidence intervals, resulting in overestimated significance level for dependent variables [18].

Software engineering data, such as various object-oriented metrics for example, are typically of count type, measured on an absolute scale, and clustered around low values [7]. This fact requires use of appropriate statistical models. For this reason the appropriate statistical models for count type of the data are negative binomial model and zero inflated models [20].

Reliability Growth Models

An important aspect of software quality is reliability. Measures of reliability widely used in Software Engineering include the number of failures discovered and the rate of discovery [16]. A Software Reliability Growth Model is a formal equation that describes the time of discovery of defects.

The literature on SRs has partially overlapped that of software reliability, since SRs often refer to occurrences of faults that also affect the reliability of software systems. Wood [26] evidences that the models used for describing software reliability can be used also for the overall analysis of SRs, without any major loss of precision.

Several reliability growth models have been proposed for software systems. Table 1 contains a synopsis of selected models. This table is an extension with modifications of Table A in [26]. Our variation of the Weibull model (W-S model) accounts more for the initial learning curve, having the S-shaped behavior more pronounced.

It is a widely accepted fact that it is not possible to select a single “best” general software reliability growth model [16]. For each project, product release, and for each goal it is necessary to select the most suitable model [23].

3 FRAMEWORK FOR EVALUATION OF PP PRACTICE

A framework for a controlled experiment is composed of

Model	Properties
GO S-Shaped (GO-S) *S-shaped*	$a(1-(1+bt)e^{-bt})$ $a \geq 0, b>0$
Goel-Okumoto (GO) *Concave*	$a(1-e^{-bt})$ $a \geq 0, b>0$
Gompertz (G) *S-shaped*	$a \cdot b^{c^t}$ $a \geq 0, 0 \leq b \leq 1, 0<c<1$
Hossain-Dahiya/GO (HD) *S-shaped*	$a(1-e^{-bt})/(1+ce^{-bt})$ $a \geq 0, b>0, c>0$
Logistic (L) *S-shaped*	$a/(1+be^{-ct})$ $a \geq 0, b>0, c>0$
Weibull (W) *S-shaped*	$a(1-e^{-bt^c})$ $a \geq 0, b>0, c>0$
Weibull *more* S-shaped (W-S) *S-shaped*	$a(1-(1+b \cdot t^c) \cdot e^{-bt^c})$ $a \geq 0, b>0, c>0$
Yamada Exponential (YE) *Concave*	$a(1-e^{-b(1-e^{-c \cdot t})})$ $a \geq 0, b>0, c>0$
Yamada Raleigh (YR) *S-shaped*	$a(1-e^{-b(1-e^{-c\frac{t^2}{2}})})$ $a \geq 0, b>0, c>0$

Table 1: SRGM models (extension with modifications of Table A in [26])

the following components: definition, plan, operations, and interpretation of the experiment [2].

The definition of the experiment is used to set up a clear motivation of the experiment and to provide details about the object, purpose perspective, domain, and scope of the experiment. Motivation of our experiment is assessment of the possible ways for improvement of the existing process and assuring high quality of the resulting products. Consequently, objects of the experiment are the development process and the resulting products. The domain and scope of the experiment, including characteristics of the development environment and the projects, should also be precisely defined.

With a precise definition of the experiment in place, it is possible to proceed to the planning phase. Planning covers issues of experiment design, criteria for comparison between the testing groups, and methods for measurement.

The motivation of our experimental framework is to understand and quantitatively assess the impact of the pair programming practice to the existing software development process in the company. This information could be then used to improve the existing process and assure high quality of the products.

The software development groups can be relatively big and geographically distributed. This brings an issue of internal organization of the group, and breaking it down to smaller teams. There should be at least two teams in the experiment, one of which is doing software development using PP approach, and the other using the existing process. Tams should have no more than 10 developers.

The characteristics of the teams performing the experiment and the characteristics of the developed projects should also be determined. The precise definition of the environment, development tools, and testing procedures has to be available for each of the teams. To determine the scope of the experiment, it is also necessary to have information about the projects and distribution of tasks between the teams.

The flowchart of the activities in the experiment is shown in Figure 1.

Criteria for assessment of the results of the new practice are based on its effects on the quality of the developed software and the costs of development. As discussed in Section 2, the different aspects of the quality and cost can be assessed using the models built on the CK metrics suite and external measures such as number and occurrence of SRs. The results of the experiment can be analyzed using framework based on multivariate statistical models applicable for the count data, software reliability growth For operation of the experiment, the preparation should provide the company developers and managers with basic training in pair programming and extreme programming concepts. Data collection can then be performed through the development of the projects. Finally, the quantitative and qualitative analyses are performed using the established framework.

4 DATA COLLECTION AND ANALYSIS

As mentioned, the set of six object-oriented design metrics from the CK metrics suite can be used to describe influence of the new development methodology to the design decisions.

Depth of inheritance tree (DIT) for a class corresponds to the maximum length from the root of the inheritance hierarchy to the node of the observed class. Another

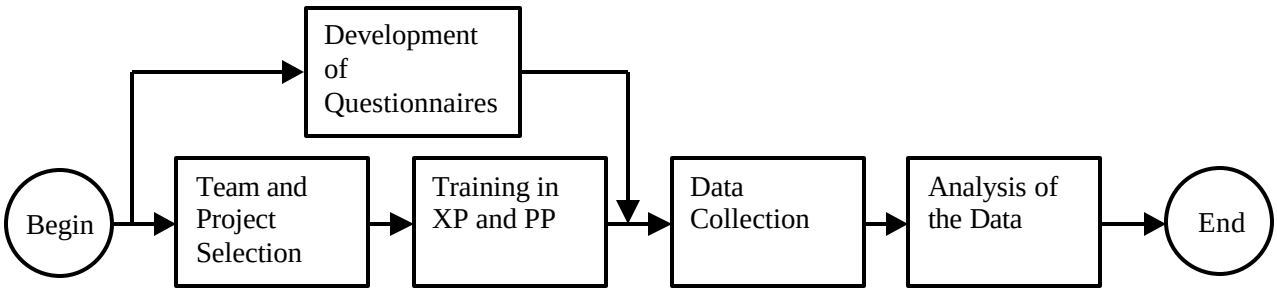

Figure 1: Process flowchart for the experiment.

metrics related to inheritance is the number of children (NOC), representing the number of immediate descendants of the class in the inheritance tree. Coupling between objects (CBO) is defined as the number of other classes to which a class is coupled through method invocation or use of instance variables. Response for a class (RFC) is the cardinality of the set of all internal methods and external methods directly invoked by the internal methods. We use number of methods (NOM) as a simplified version of more general weighted methods count (WMC), as usually done [2]. The number of internal methods is extracted instead of forming a weighted sum of methods based on complexity. The lack of cohesion in methods (LCOM) is defined as the number of pairs of non-cohesive methods minus the count of cohesive method pairs, based on common instance variables used by the methods in a class.

In addition to CK metrics, the count of Lines Of Code (LOC) can be easily collected. This information can be used to determine if simpler models, based on size metrics, could have the same explanatory power. For code written in C++ and Java, source lines of code can be counted using semicolons.

The dependent variable in analysis, number of defects or SRs, is count variable measured on an absolute scale, and clustered around low values [7]. The appropriate statistical models for count type of the data are negative binomial model and zero inflated models [20], derived from the Poisson distribution.

The most common distributions applied to count data are based on the Poisson and multinomial distributions [19]. The Poisson distribution is particularly suitable for counting events occurring over time. In the corresponding Poisson Regression Model (PRM), the Poisson distribution determines the probability of a count, where the mean of the distribution is a function of the independent variables. PRM has been used in software engineering for modeling the number of faults [13] and the effort expressed in hours [6].

PRM requires equidispersion, i.e., equality of the conditional variance and the conditional mean of the dependent variable. When conditions for the PRM are not met, e.g., in case of high conditional variance of the dependent variable, the Negative Binomial (NB) distribution and the associated NB Regression Model (NBRM) can be used [19,6].

It is common in software metrics data that the number of zeros exceeds the prediction of both PRM and NBRM. Zero-inflated count models explicitly model the number of predicted zeros [17].

Software Reliability Growth Models (SRGMs) give an additional insight into the defect behavior of the software product and the effort necessary for achieving the desired quality.

The parameters of the SRGMs can be estimated using least square error regression on the available service request (SR) data. The statistical tool that we use to tune the model parameters employs an iterative estimation algorithm for finding the global minimum of the cost function. We also use the bootstrap method is used to determine the confidence intervals for parameters of models.

The goal of modeling the occurrences of SRs in this paper is to create an accurate description for assessment, comparison and improvement.

This goal can be further organized in terms of goodness of fit, the accuracy of the final point, relative precision of fit, and coverage of fit.

The goodness of fit represents how well the model fits the data, and therefore it is a reliable descriptor of the overall process, to be used for comparison and assessment.

The accuracy of the final point represents whether the model is able to determine the total final number of SRs. It is measured with E:

$$E = 100 \cdot \left| \frac{A - a}{A} \right|$$

where A and a are respectively the true and estimated value of the total SRs served.

The relative precision of fit is the size of the bootstrap 95% confidence interval computed over the parameters of the model and normalized over the size of the interval of time of SRs arrival.

The coverage of fit is the degree to which the 95% confidence interval captures the oncoming service

requests (shown in the Data Coverage column of Table 2).

Relative precision of fit and coverage of fit measure two complementary aspects of the fit that must be considered together to evaluate the value of a model: a very large 95% confidence interval might be able to capture most of the data, but it would be totally useless.

There are two major classes of mathematical functions representing SRGMs with different defect-detection rates: concave and S-shaped. S-shaped models are first convex and then concave, with a period during which the error-detection rate increases, reflecting the initial learning phase, that is, the assumption that later testing is more efficient than early testing [21].

The observed shape in occurrence of SRs and consequently the SRGMs that describe this process provide a way to compare the traditional development process with the new methodology being introduced. Potentially increased learning rate in pair programming would result in shorter initial concave period and steeper defect detection rate.

5 CONCLUSION

Some anecdotal evidence argues success of the XP in producing higher quality software in less time.

It was our goal in this paper to set-up a framework and measurement plan for objective assessment of PP practices introduced to selected part of a company.

Our goals were also to provide insight into impact of the new practices to the developer satisfaction and confidence and related managerial issues.

In order to assess benefits and costs of the pair programming practice we compare design aspects of the resulting software products and their defect behavior using the CK metrics suite. In addition, we use software reliability growth models to describe the differences between the two processes with respect to occurrence of service requests and effort necessary for fixing them.

The work based on the experimental framework described in this paper is currently in progress in cooperation with a major North-American telecommunication company. We will provide experimental results as soon as they become available.

REFERENCES:

1. Basili V.R., Briand L.C., Melo W.L. "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, 22(10), 1996
2. Basili V.R., Selby R.W., and Hutchens D.H. "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, 12(7), July, 1986
3. Basili V.R., Weiss D.M. "A method for collecting valid software engineering data" *IEEE Trans. on Software Engineering*, 10(6), 728-738, 1984
4. Beck K. *Extreme Programming Explained: Embrace Change*, Addison-Wesley Pub Co, 1999
5. Briand L.C., El Emam K., and Morasca S. "On the Application of Measurement Theory in Software Engineering," *Journal of Empirical Software Engineering*, 1(1), 1996
6. Briand L.C. and Wüst J. "The impact of Design on Development Cost in Object-Oriented Systems," Technical report, *http://www.iese.fhg.de/network/ISERN/pub/technical_reports/isern-99-16.pdf*, 1999
7. Chidamber, S.R., Darcy D.P., Kemerer C.F. "Managerial Use of Object-Oriented Software: An Explanatory Analysis," *IEEE Transactions on Software Engineering*, 24(8), 1998
8. Chidamber S.R. and Kemerer C.F. "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, 1994
9. Cockburn A. and Williams L. "The Costs and Benefits of Pair Programming," *eXtreme Programming and Flexible Processes in Software Engineering XP2000*, 2000
10. eGroups email group service http://www.egroups.com/message/extremeprogramming, 2000
11. Fenton N.E. and Neil M. "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering*, 25(5), 675-689, 1999
12. Fenton N.E. and Pfleeger S.L. *Software Metrics: A Rigorous and Practical Approach*, Brooks/Cole Pub Co, 1998
13. Graves T., Karr A.F., Marron J.S., Siy H. "Predicting Fault Incidence Using Software Change History," *IEEE Transactions on Software Engineering*, Vol. 26, No. 7, July, 2000
14. Gray A.R. and MacDonell S.G. "A comparison of techniques for developing predictive models of software metrics," *Information and Software Technology*, 39, 425-437, 1997
15. Humphrey W.S. *Managing the Software Process*, Addison-Wesley Pub, 1989
16. Mendonça M. and Basili V. "Validation of an Approach for Improving Existing Measurement Frameworks," *IEEE Transactions on Software Engineering*, Vol. 26, No. 6, pp. 484-499, June, 2000
17. Lambert D. "Zero-inflated poisson regression with an application to defects in manufacturing," *Technometrics*, 34, 1-14, 1990

18. Littlewood B. “Stochastic Reliability Growth: A Model for Fault Removal in Computer Programs and Hardware Design,” *IEEE Transactions on Reliability*, Dec. pp.313-320, 1981

19. Lloyd C.J. *Statistical Analysis of Categorical Data*, Wiley-Interscience, 1999

20. Long J.S. *Regression Models for Categorical and Limited Dependent Variables*, Advanced Quantitative Techniques in the Social Sciences, No 7, Sage Publications, 1997

21. Lyu M.R. *Handbook of Software Reliability Engineering*, McGraw Hill, 1996

22. Succi G., Pedrycz W., Stefanovic M., Paynter G., Sloane D. “Empirical Analysis of Pre-Release Service Requests for Real Time Software Systems”, *Technical report QUASE-TR-2000-01*, Quantitative Software Engineering Lab, University of Alberta, 2000

23. Succi G., Stefanovic M., Pedrycz W., Musilek P. “Predicting Software Service Requests,” *ASERC Workshop on Quantitative Software Engineering*, Banff, 2001

24. Williams L., Kessler R.R., Cunningham W., and Jeffries R. “Strengthening the Case for Pair Programming,” *IEEE Software*, Vol. 17, No. 4, July/August, 2000

25. Williams L. “The Collaborative Software Process,” PhD Dissertation, http://www.cs.utah.edu/~lwilliam/Papers/dissertation.pdf, 2000

26. Wood A. “Predicting Software Reliability”, *IEEE Computer*, Vol. 29, No. 11, November, pp. 69-77, 1996