An Initial Exploration of the Relationship Between Pair Programming and Brooks' Law

Laurie Williams[1], Anuja Shukla[2], Annie I. Antón[1]
Department of Computer Science, North Carolina State University
[1]{williams, anton}@csc.ncsu.edu
[2]anuja@idealake.com

Abstract

Through his law, "adding manpower to a late software project makes it later," Brooks asserts that the assimilation, training, and intercommunication costs of adding new team members outweigh the associated team productivity gain in the short term. Anecdotes suggest that adding manpower to a late project yields productivity gains to the team more quickly if the team employs the pair programming technique when compared to teams where new team members work alone. We utilize a system dynamics model which demonstrates support of these observations. Parameter values for the model were obtained via a small-scale, non-probabilistic, convenience survey. Our initial findings suggest that managers should incorporate the pair programming practice when growing their team.

1 Introduction

Documented in 1975, Brooks' Law asserts "adding manpower to a late software project makes it later." [7] Brooks' advice is a debilitating statement to project managers who increase manpower in an attempt to salvage a late project. Brooks stresses this is akin to "dousing a fire with gasoline" [7] because adding manpower increases training costs, assimilation time (time lost learning about the project) and intercommunication overhead (work required to communicate, formally or informally, among the team members). Only when increased effective manpower outweighs these costs will the addition of new team members lead to an improvement in the completion date.

Organizations shared anecdotal claims of experiencing non-Brooks' Law phenomenon when they utilized pair programming. As a result, we examined several system dynamics models that aimed to explain project effects predicted by Brooks' Law. We selected one developed by Stutzke [29] to demonstrate the potential effects of pair programming. We obtained parameter values for the model via an email survey and utilized the resulting model to examine the relationship between Brooks' Law and pair programming.

The remainder of this paper is organized as follows. Section 2 provides some exemplar anecdotes of teams that utilized collaborative practices in their growth strategy; these types of anecdotes motivated this research. Section 3 provides an overview of related and relevant work. Section 4 discusses Stutzke's model and case study. Section 5 presents our survey results, and Section 6 applies these results to Stutzke's model. Finally, Section 7 summarizes our findings and future work.

2 Motivation

Teams have anecdotally reported success growing teams by using pair programming. For example, one survey-based study reported that since pairing is a part of daily life, team members do not have to stop working to help out the new person. Much of the mundane technical training can be assimilated as part of the job [28]. Additionally, we share anecdotes of software development projects at Motorola and Menlo Innovations in which team members practiced collaborative techniques, including pair programming.

2.1 Motorola

Figure 1 portrays much of the history of an integration project at Motorola that employed Extreme Programming (XP) [4] for its development paradigm, eventually using all core practices [34]. From the start of the project, the team practiced short (two to three weeks) iterations, the planning game, and pair programming. The task was to integrate their internal Java implementation of Diameter with CMU Monarch Mobile IP. In XP projects, teams measure their progress via project velocity using unit of effort called *ideal time*. Ideal time is time without interruption where one can concentrate on work and feel fully productive [5]. The Motorola team calls ideal time Ideal Engineering Days (IED) and measures their velocity metric by IEDs/work day.

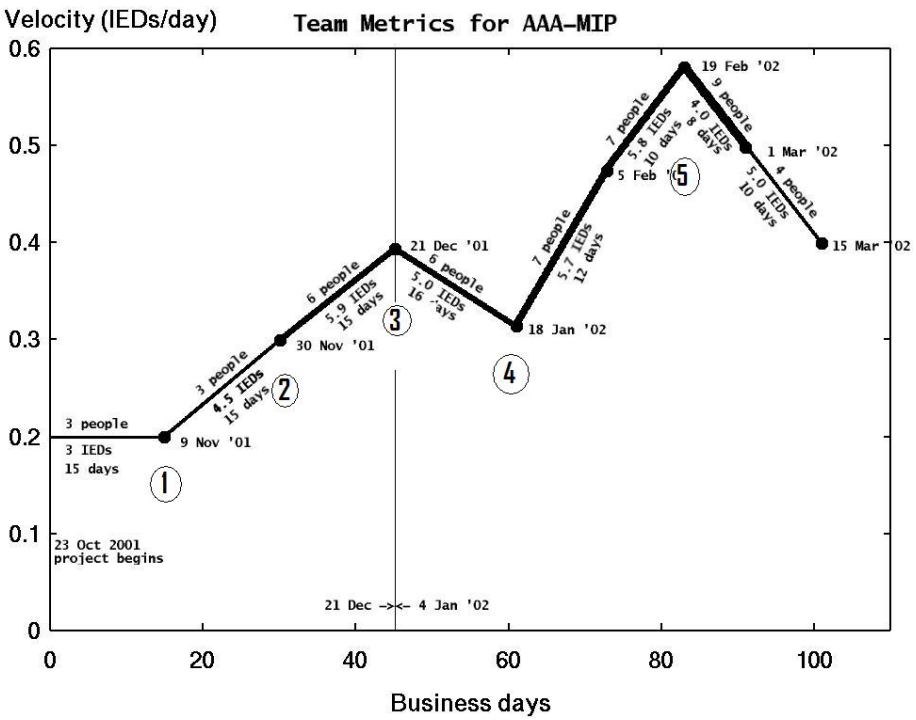

Figure 1: Motorola Project History (adapted from [34])

Initial estimates suggested the team might have difficulty meeting the targeted deadline with the three people initially assigned to the project. However, the team waited until the end of the first iteration (Point 1 on Figure 1) to ask for help. With less than three months until project completion, management responded by providing three additional team members who started at the beginning of the third iteration (Point 2 on Figure 1). Each of the new team members paired with one of the existing team members. The team expected a Brooks' Law slowdown for at least one iteration. They were pleased that they did not experience such a slowdown. Instead they maintained their initial acceleration. They also added new team members at several other points in the project while maintaining acceleration, as shown in Figure 1.

The team believed that the slow-down after the holidays (between Point 3 and Point 4 on Figure 1) was simple post-holiday malaise. There were several changes after the holidays that improved collaboration and communication. They moved their offices from separate cubes to a single shared space. Their customer representative started working with them more often and finally began to sit with them for the fifth iteration (starting at Point 4 on Figure 1). The team's acceleration between Point 4 and Point 5 was almost exactly twice the acceleration experienced from Point 1 to Point 3. During this time, they were practicing pair programming and working in a common collaborative environment shared by a participatory customer.

Halfway through the iteration started at Point 5, management told the team that the project would end early and that they would remove five people from the project. This accounts for the sharp decline in velocity. The velocity is only approximate for the last iteration because the team created and destroyed story cards throughout the iteration.

2.2 Menlo Innovations

The founders of Menlo Innovations faced a different kind of challenge at a client site [12, 13]. Marketing demand required that the team quickly grow from 14 to 34 people. Figure 2 depicts the Menlo Innovations project history. The productivity values shown in this figure illustrate relative productivity per time period (person days) and are not exact values. After working on a project with 14 developers for six months using the XP process, the manager was asked to hasten the progress of the project by more than doubling the team size. As shown in Figure 2, in two months (between Points 2 and 4) almost 20 new programmers joined the team.

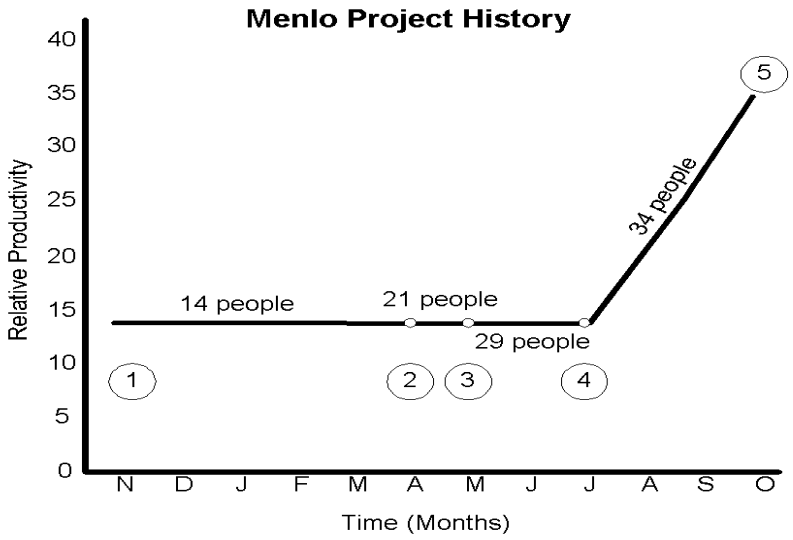

Figure 2: Menlo Project History

With each addition of new team members, the manager broke apart existing pairs and paired new persons with an existing employee possessing strong mentoring skills. Each two weeks, pairs were reassembled. After just six weeks, new team members mentored programmers just joining the team. This was possible because of the strong working relationships formed by pairing in the previous six weeks. The mentoring was further facilitated by an open, collaborative workspace where access to those people and others allowed free flow of questions and answers.

During the three months following the recruiting effort, (between Points 2 and 4) the augmented team continued to produce approximately the same level of results as the earlier team, hence the flat relative productivity. That is to say, a team of 34 was producing the same amount of work as a team of 14. They were not producing less in spite of the significant cross-training and mentoring that was required. During this period, there was a decrease in quality of the work product, but it was detectable and correctable, given strong XP unit testing practices [6]. Then, the team began experiencing a nearly linear increase in productivity based on the increase in team size. The team noted that cross-training and mentoring was bi-directional as the new team members were bringing as many new skills and experiences to the pairing relationship as those on the original team.

To summarize, both teams faced staffing challenges. The teams were able to more than double their team size without a decline in productivity. Both teams indicated that pair programming was an essential enabler that prevented them from suffering from the Brooks' Law phenomenon. Their experiences motivate our study.

3 Background and Related Work

This section provides an overview of pair programming, Brooks' Law, systems dynamic models of Brooks' Law, and survey-based research.

3.1 Pair Programming

Previous research with senior-level undergraduate students showed that pairs developed higher quality code in about half the time as solo programmers [31, 33]. The higher quality code supports Harlan Mills' belief that "programming should be a public process" [7]. Exposing programs to other programmers helps quality control due to the constant peer pressure to do things well and because peers spot flaws and bugs [7].

Most anecdotes and industry case studies support the results of this study [32]. However, a similar experiment [19] has concluded contradictory results, indicating that pairs spend almost twice as much total programmer effort as solo programmers. Examining their study, pairs did take twice as much time in the first program that took the students between 2.5-4 hours to complete. These findings are consistent with earlier studies [20, 33] indicating that pairing is significantly more expensive when programmers are first learning the dynamics of pairing. By the end of the fourth program in their study, students had about 13 hours of pairing experience. In the fourth program, the pairs were beginning to look more affordable. Our observations, as well as anecdotal evidence, suggest that it takes programmers a few hours to a few days to transition from solo to collaborative programming [31].

Pair programming offers additional benefits, including:

- *Increased Morale.* Pair programmers are more satisfied with their work arrangements [32, 33].
- *Increased Teamwork.* Pair programmers get to know their teammates much better because they actively collaborate [32].
- *Increased Knowledge Transfer*. Pair programmers transfer their knowledge to their partners as a normal part of collaborating [21]. As a result, there are always at least two developers who are knowledgeable about a particular piece of code. With pair rotation, often more than two team members understand a particular piece of code. The term *pair rotation* [32] is used to denote when programmers pair with different team members for varying times throughout a project. Ideally, a team member pairs with the person who can most help them on a particular task.
- *Enhanced learning.* Pairs continously learn by watching how their partners approach a task, how they use language capabilities, and how they use development tools [8, 32].
- *Enhancing knowledge creation*. Pair programming leads to more effective knowledge creation because it forces the developers to explicitly articulate their thoughts. It also stimulates continuous feedback by the partner [16].

3.2 Brooks' Law Factors

Brooks' Law is based upon the following three cost factors [7]:

Increased training costs. In the software industry, new team members are generally trained by a mentor who is familiar with project details and who provides guidance about how to be effective and successful in the organization. During this training time, the mentor must reallocate his or her own valuable time away from his or her own project responsibilities. The cost of the mentor's time varies linearly with the number of new team members.

Increased assimilation time. This is the time required for a new team member (who already possesses the necessary background knowledge and skills) to become an effective, contributing team member. Assimilation time includes the time necessary for the new team member to understand project-specific facts, such as facility layout, policies, procedures, project domain, and the development and test environment. This cost also varies linearly with the number of new team members.

Increased intercommunication overhead. Inter-communication overhead is defined as the average team member's drop in productivity below his nominal productivity as a result of team communication [2]. Intercommunication time includes verbal communication, documentation, and any additional work required for formal or informal communication among team members. This cost increases non-linearly (N^2), as more people are added to a team.

3.3 System Dynamics Models

Brooks' Law is often studied from a system dynamics perspective. These models provide valuable information, yet are limited in their ability to accurately predict the outcome of actual software projects. The models can produce the illusion that project management is deterministic, when in fact no human process, particularly software development, is deterministic. Abdel-Hamid classifies software development as a multiloop, nonlinear feedback system. These types of systems produce puzzling behavior, requiring more than intuitive judgment alone for estimation [3]. As a result he, Forrester [11] and others have demonstrated the feasibility and utility of studying "computer-based microworlds" to learn about complex social systems such as software development. Several such models will now be discussed.

In 1977, Gordon and Lamb studied Brooks' Law's three factors (training, assimilation and intercommunication). They advocated adding more than the expected number of people to a project early on and then not changing anyone's tasks or responsibilities until the project is completed [14]. Abdel-Hamid and Madnick developed a quantitative model of project dynamics to study Brooks' Law in 1991. They concluded that adding people to a late project will not always cause it to be late, but will always cause the project to overrun its budget [1, 2]. The Abdel-Hamid and Madnick model spurred the development of other similar models [17, 25, 26]. More recently in 1992, Weinberg argued that training and communication coordination overhead are the two factors that most contribute to Brooks' Law because an increase in both factors manifests itself as added work [30].

In 1994, Stutzke observed that Brooks' Law does not always hold true [29], contending that under certain conditions, manpower can be added to a late project to meet a specified product delivery date (or even accelerate the delivery date). He developed a simple model to determine the conditions under which adding staff will benefit a project and to predict the amount of additional useful effort delivered to a project. Stutzke's model considers (1) the time required for a new team member (who possesses the necessary background knowledge and skills) to be assimilated into the workforce; (2) the time remaining to complete the project; and (3) the amount of mentoring the existing

staff must provide to a new team member. This model is detailed in Section 4.

In 1999, Hsia et al. argued that Abdel-Hamid and Madnick based their assertion on two unrealistic assumptions: that development tasks can be partitioned without consideration of the sequential constraint (e.g. adding more people will not help if tasks must be completed sequentially) and that when project managers sense a shortage of manpower they will continuously add new people to a project [15]. Consequently, Hsia et al. incorporate the sequential constraint and assume that people are only added to a project once during a project lifecycle. They found that although adding people to a late project always increases its cost, the project may not always be late [15]. According to Hsia et al., every project has a critical point in its schedule, *T,* typically about one-third of the way through the schedule. If enough manpower is added before *T*, the project can still finish before its scheduled deadline. Alternatively, if manpower is added after *T*, then the project will be late.

None of the models discussed above consider the effects of intercommunication overhead or the nontrivial process of repartitioning. Stutzke contends that assessing the effect of intercommunication overhead on project schedule is challenging because communication is a second-order effect. It is therefore not possible to quantify its effects on the project schedule directly [29].

The quantitative models discussed above are used to study project dynamics. Instead, we use these models to study the potential of pair programming to mitigate the costs of bringing new team members into a project; Stutzke's model supports this kind of analysis and serves as the basis for our investigation.

3.4 Survey-Based Research

A survey is a retrospective study of a situation to document relationships and outcomes [10]. When comparing surveys to experimentation and case studies, surveys have the least control on variables and therefore suffer from internal validity issues. However, people from different, real-world contexts are involved in surveys, leading to external validity strengths and the result are more generalizable.

The ability to collect large amounts of data without large cost makes electronic surveys, such as email-based surveys, attractive [23]. Email surveys are conducted by sending an email with a questionnaire as an attachment or by including in the note itself. Respondants open the survey, fill it in, and send it back. Email surveys are not anonymous.

Sampling is the process of selecting a set of respondants as a subset of the entire population under study. Probabilistic sampling is a systematic approach to select the sample so that every member of the population has an equal chance of being selected. The advantage of this procedure is that it is possible to derive unbiased, representative results from the sample that hold for the entire population with only a low sampling error [18].

Non-probabilistic sampling is used when systematic sampling is not possible or in small-scale surveys. One means of non-probabilistic sampling is for the reseracher to choose convenient persons to act as respondents. Convenience samples have the risk of being biased [23] because the people who are willing to participate may differ in important ways from those who are not willing. Therefore, the results reported via non-probabilistic, convenience-sample surveys must be qualified based upon the respondent population [18]. The survey conducted as part of this research was such a survey.

4 Stutzke's Model for Adding Team Members

Mathematical models, such as those discussed in Section 3.3 help estimate the effects of adding new team members to a project that is already behind schedule. These models consider factors such as training and assimilation time before a new team member can make a positive contribution to overall team productivity. They offer several alternatives to adding new team members to meet the project deadline. Stutzke's model, in particular, is especially well suited for demonstrating the impact that pair programming can have on the training and assimilation factors influencing Brooks' Law.

The purpose of Stutzke's research was to develop and validate a mathematical expression for Brooks' Law. Stutzke performed his research in three steps. First, he developed equations for his model. Second, he administered a survey to obtain values for assimilation time (a) and mentoring time (m). Other project-specific data collected via the survey provided the data for a case study. Finally, he validated his model through an industrial case study.

In the case study, a software development project that had a specified completion date fell behind schedule. Additionally, the project manager knew the remaining work and the net effort needed to finish the project. The manager decided to add more team members to accomplish the required tasks by the specified completion date. The manager needed to predict the total effort and duration needed to complete a project, considering training and assimilation.

The case study validated that Stutzke's research provided a valuable quantitative model for helping the manager to determine the following:

- the useful effort delivered by the total staff as a function of the number of people added;
- the maximum number of new team members that can be added; and
- how late team member size can be increased and still produce a net gain.

In our research, we employed the most relevant parts of Stutzke's model to examine the potential of pair programming within late software projects. We now provide a brief description of these parts.

4.1 Notation

Stutzke uses the following notations in his model's equations [29]:

N_i = initial number of staff
N_f = final number of staff
f = fractional increase in staff = $(N_f - N_i) / N_i$
r = remaining time (workdays) to complete the project from the day additional team members arrive
m = training cost i.e. the fraction of staff member's time spent mentoring one new person
a = individual assimilation time i.e. the number of workdays the new team member spends learning the project
a′ = the effective assimilation time, which includes the time spent by both the mentor and the trainee = a * (1+m)
E_u = useful effort delivered to project (person days)
E_{gain} = net gain in effort with the augmented staff (person days)

4.2 Assumptions

The labour costs are assumed to be the same for both the trainee and mentor, which allows the combining of the trainee and mentor's effort to obtain *a*, the assimilation time. Figure 3 shows the useful effort delivered to the project's tasks before, during and after assimilation takes place.

Before assimilation, $E_u = N_i$.

During assimilation, $E_u = (1 – f * m) * N_i$. This takes into account the proportion of time that some of the initial staff members spend mentoring rather than producing results for the project. Thus, the project will see an initial drop in the rate at which the useful effort is delivered to the project's tasks.

After assimilation, $E_u = N_f$.

Once all the team members are assimilated, useful effort will be delivered at a higher rate. As we observe from Figure 3, there is ultimately an increase in the delivery rate.

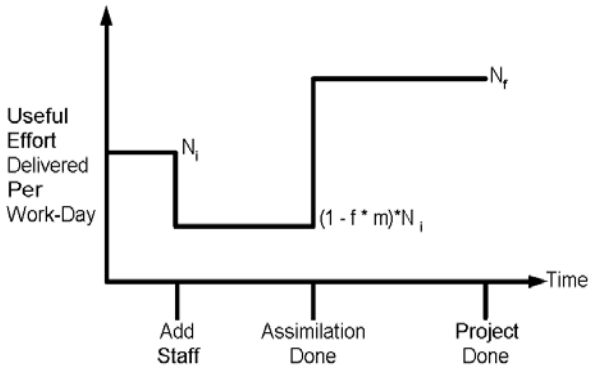

Figure 3: Delivery Rate Versus Time (adapted from [29])

The useful effort delivered to the project is the effort applied to perform the tasks needed to complete the work. We represent this as the sum of work delivered during the mentoring period (duration *a*) and the work delivered after new team member assimilation. The useful effort delivered over the entire time interval *r* is:

$$E_u = N_i*(1-f*m)*a+(1+f)*N_i*(r-a)$$
$$= f * N_i * (r-a') + N_i * r$$

[where a′= a * (1+m), as discussed above]

Stutzke's model examines the trade-off between the number of people added to the project and the amount of useful output delivered to the project. Managers should not add staff unless there is a net gain in a team's output after the cost of adding new team members is included. The net gain in effort provided to a project is the difference between the useful effort provided by the augmented staff, E_u, and the original staff's potential total effort:

$$E_{gain} = E_u - (N_i * r)$$
$$= f*N_i*(r-a') + N_i*r - N_i*r$$

[substituting E_u from previous equation]

$$= f * N_i * (r-a')$$

The above equation demonstrates that there is a net gain if $f > 0$ and $r > a'$. The condition $(f > 0)$ means that managers must add some additional team members. The condition $(r > a')$ means that the project must have enough time remaining to assimilate the new team members.

We interpret breakeven graphically from Figure 3; the area under the curve represents the breakeven work delivered (rate * duration). To produce a net gain in effort, the effort provided by the additional staff (after assimilation) must be more than the effort lost during assimilation.

4.3 Case Study

Stutzke utilized these equations in a case study of an industrial software development project with a Software Engineering Institute Capability Maturity Model (SEI CMM) Level 3 organization [22]. The conditions in the case study were as follows:

- This project was considered to be approximately half complete.
- The effort needed to complete the project could be estimated, since the following were well known: the software architecture, all component modules, their sizes, and the needed modifications.
- There was an initial staff of ten software engineers and testers.
- The time remaining to complete the project on schedule was 5.5 calendar months (about 121 workdays).
- All staff would work on average of 15% overtime.
- The effort remaining was about 20,000 person hours.

Using Stutzke's expressions, we estimate how many new team members need to be added to complete the project as per schedule. The original staff would be able to deliver the following effort during the 121 work days without any additional help:

$E_i = N_i * (1+ \text{Overtime}) * r * 8$

$= 10 * 1.15 * 121 * 8$

$= 11{,}132$ person hours

The new team members must thus deliver an additional 8,868 person hours (20,000 – 11,132). We refer to this as E_{add} and utilize this value in Section 6.

5 Surveying for Model Parameters

We designed two surveys (Survey I and Survey II) to investigate the effects of pair programming on training costs, assimilation time, and to obtain other important contextual information. We administered these surveys to professional software developers to obtain values for our model-based analysis of the relationship between pair programming and Brooks' Law. Both surveys were "case control" [18] because the respondents were asked about previous circumstances to help explain the phenomenon under study.

In the fourth quarter of 2001, the two surveys were emailed to 78 individuals who had previously answered a web-based survey compiled as research for *Pair Programming Illuminated* [32]. Additionally, the survey questions were posted on an Extreme Programming [4] (XP) message board[1]. In both cases, respondents returned the survey to the first author via email.

We sought to obtain responses from project managers and from senior- and mid-level software developers with experience in any programming language or platform. Our selection criteria eliminated from analysis any respondents that did not have five years of programming experience in both pair and solo programming environments. Because most software engineers cannot meet this criteria, our sample was very specific and of limited availability, necessitating a non-probabilistic sample. Additionally, our sample was chosen primarily from those who practice XP since this is the largest population that could meet our criteria. This sample may not be representative of the general software engineering population. Only 30 respondents answered each survey, resulting in a small-sample population. Finally, the answers to the survey obtain subjective information from the respondents. As a result of these limitations, we view our study as directional and anecdotal. We utilize the results to examine the relationship between pair programming and Brooks' Law to explain anecdotal observations of industrial teams. We do not attempt to draw any statistical inferences from our samples.

Survey I, included in Appendix A, was related to pair programming and training. This survey was adapted from a similar, validated survey administered by Stutzke [29] for his research on a mathematical expression for Brooks' Law. Survey II, in Appendix B, examined pairing combinations and pair rotation.

5.1 Survey I: Assimilation and Training

The 30 responses received for Survey I met our selection criteria. This survey consisted of two open-ended questions. Respondents were asked to estimate three values for both assimilation time (*a*) and mentoring time (*m*) with and without pairing when new members are added to the team. The three estimated values were their perceived "Low", "Most Likely", and "High" values. We then determined the average value for *a* and *m* [27] shown in Table 1 using the Program Evaluation and Review Technique (PERT) formula [24], which is often used for determining best estimates:

PERT average = (lowest + 4*most likely + highest)/6

Table 1: Survey values for assimilation and mentoring

	With Pairing	Without Pairing
Assimilation (a) (total days)	12	27
Mentoring (m) (% of day)	26%	37%

[1] http://groups.yahoo.com/group/extremeprogramming/ with over 3000 subscribers.

Without pair programming mentoring can be considered to be lost time for the mentor [9]. However, with pairing new team members help their mentor during the training process by asking questions and by identifying defects in their work. In this way, new team members are contributing [31]; the mentor is slowed from making progress but the time is not lost.

5.2 Survey II: Pairing Combinations and Pair Rotation

After obtaining the results of Survey I, we desired additional information to provide enriched information to managers who might use pair programming for ameliorating late projects. Survey II was sent and posted to the same audience as Survey I and consisted of two questions. The questions focused on pairing combinations (the mix of experience level of each member of the pair) and pair rotation. We received 35 responses. Of these, 15% were from individuals who had also participated in Survey I. The remaining responses were from new individuals. Only 30 respondents met our survey experience criteria.

5.2.1 Pairing Combinations. In Survey II, respondents indicated the typical development experience of new person's mentor. The results are shown in Figure 4 [27]. According to the respondents, mentors with more than five years of experience are used much more often than inexperienced mentors are used. These results indicate that managers who utilize pair programming for growing their team should utilize experienced mentors as possible. Pairing a new team member with an experienced partner accelerates the new team member's learning curve. However, pairing a new team member with another new team member has been found to be better than leaving that person alone [32]; the Menlo team discussed above paired new people with relative novices.

5.2.2 Pair Rotation. The next survey question sought to develop an understanding of the teams' use of pair rotation. The results indicate that pair rotation is widely practiced; 94% of developers cited its use [27]. As a result, it must be considered that the analysis herein is predicated on the practice of dynamic pairing (via pair rotation), not the use of static pairing whereby two team members pair for long period of time.

6 Stutzke's Model and Brooks' Law

We will combine Stutzke's model and values based upon our survey results. In our survey, we obtained values for assimilation (*a*), mentoring (*m*) as shown in Table 1. We can use these values to compute effective assimilation (*a′*), as shown in Table 2:

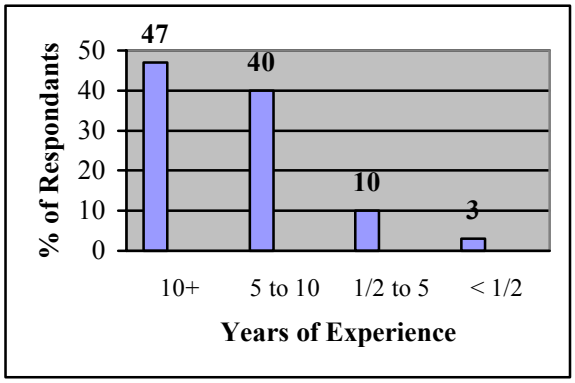

Figure 4: Experience level of new team members' pairing partner

Table 2: Summary of Survey Values

With Pairing	Without Pairing
a = 12 workdays	*a* = 27 workdays
m = 26%	*m* = 36%
a′ = a (1+m) = 12 * (1+ .26) = 15 workdays	*a′* = a (1+m) = 27 * (1+ .36) = 36 workdays

To compare increasing a team with and without pairing, we use Stutzke's case study, as outlined in Section 4.3. The additional effort (E_{add} which was found to be 8868 in the case study) is divided by the overtime effort (15%) that will be expended in the remaining days. As shown in Table 1, the effective assimilation time *a'* without pairing is 36 workdays and 15 workdays with pairing. The fractional increase in staff is given by *f*.

f $= E_{add} / (N_i * (1+ \text{Overtime})* (r\text{-}a')*8))$
$= 8868 / (10 * 1.15* (121\text{-}a')*8)$
f (with pairing) = 0.91
f (without pairing) = 1.13

According to the model, a project manager needs to add 9 new team members if the team practiced pair programming (10 * 0.91 = 9 new team members) to complete the project on time, or 11 new team members if the team did not practice pair programming (10 * 1.13 = 11 new team members). In either case, the model shows that new team members can help complete the project on time. However, the team can save the expense of two new team members if the team utilizes pair programming, at a savings of approximately $56, 000[2].

[2] A systems analyst II in Research Triangle Park currently makes an average of $57,672 or $231/day. $56,000 is the cost of two systems analysts for 121 days each.

Our results show that the effective assimilation time and mentoring time is reduced due to pairing. Since assimilation is faster, more work can be achieved in the remaining days because (*r-a'*) is a larger number. The net gain in effort, E_{gain}, is given as $f * N_i * (r-a')$. This equation implies that breakeven ($E_{gain} = 0$) occurs much earlier because effective assimilation time is reduced with pairing.

7 Conclusions and Future Work

Software projects continue to be under pressure to deliver quickly. As a result, managers still consider alternatives of how to deal with schedule pressures and late software projects. Anecdotal observations led us to believe that pair programming can help in these unfortunate, but unavoidable, situations. Through our research, we examined the potential of the pair programming practice for helping late projects.

We sent two surveys to information technology professionals, primarily those who practice XP, to examine the effects of pair programming on the factors affecting Brooks' Law. We analyzed 30 survey responses. Using these results, we examined the effects of pairing on Brooks' Law using Stutzke's mathematical model. This model analyzed the process and costs of assimilating new team members, including the costs associated with the diversion of his or her mentor from the project task itself.

Through these initial results, we provide advice to project managers who are dealing with late software projects or the need to rapidly increase a team. Adding manpower to a late project will yield productivity gains to the team more quickly if the team employs the pair programming technique. Based upon our survey results, these organizations should also practice pair rotation, which can ease the training/mentoring burden and which can allow new team members to learn more about the overall project. Additionally, often new software engineers are paired with more senior, experienced team members. We plan to replicate the survey aspect of this study using probabilistic techniques to remove the sampling bias inherent in this initial study.

8 Acknowledgements

Many thanks to La Monte H.P. Yarroll for sharing the Motorola report and and to Richard Sheridan of Menlo Innovations for sharing his XP experiences. We thank the NCSU Software Engineering reading group for reviewing drafts of this paper and, specifically, Michael Gegick for creating Figures 2 and 3 and Nachiappan Nagappan for the cost savings example.

References

[1] T. Abdel-Hamid, "The Dynamics of Software Projects Staffing: A System Dynamics Based Simulation Approach," *IEEE Transactions on Software Engineering*, vol. 15, no. 2, pp. 109-119, 1989.

[2] T. Abdel-Hamid and S. Madnick, *Software Project Dynamics: An Integrated Approach*: Prentice Hall, 1991.

[3] T. Abdel-Hamid, "The Slippery Path to Productivity Improvement," *IEEE Software*, vol. 13, no. 4, pp. 43-52, 1996.

[4] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, Mass.: Addison-Wesley, 2000.

[5] K. Beck and M. Fowler, *Planning Extreme Programming*. Reading, Massachusetts: Addison Wesley, 2001.

[6] K. Beck, *Test Driven Development -- by Example*. Boston: Addison Wesley, 2003.

[7] F. P. Brooks, *The Mythical Man-Month*: Addison-Wesley Publishing Company, 1995.

[8] T. Chau, F. Maurer, and G. Melnik, "Knowledge Sharing: Agile Methods vs. Tayloristic Methods," IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), Linz, Austria, 302-308, 2003.

[9] T. DeMarco, "Human Capital, Unmasked," in *New York Times*. New York, 1996, pp. F13.

[10] N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*: Brooks/Cole, 1998.

[11] J. Forrester, *System Dynamics and the Lessons of 35 Years*. Cambridge, Mass.: MIT, 1991.

[12] C. J. Goebel, "Extreme Programming Used to Establish the Culture of a High Performance Team," http://www.menloinstitute.com/freestuff/whitepapers/Management%20Case%20Final.pdf, no., 2002.

[13] C. J. Goebel, T. Meloche, and R. Sheridan, "Extreme Interviewing," http://www.menloinstitute.com/freestuff/whitepapers/Extreme%20Interviewing%20Final.pdf, 2002.

[14] R. L. Gordon and J. C. Lamb, "A Close Look at Brooks' Law," *Datamation*, vol. June, no. pp. 81-86, 1977.

[15] P. Hsia, C. Hsu, and D. C. Kung, "Brooks' Law Revisited: A System Dynamics Approach," International Computer Software and Applications Conference (COMPSAC '99), 370-375, 1999.

[16] T. Kähkönen and P. Abrahamsson, "Digging into the Fundamentals of Extreme Programming," 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03), Belek-Antalya, Turkey, p. 273-280, 2003.

[17] M. I. Kellner, R. J. Madachy, and D. M. Raffo, "Software process simulation modeling: why? what? how?," *Journal of System and Software*, vol. 46, no. pp. 91-105, 1999.

[18] B. Kitchenham and S. L. Pfleeger, "Principles of Survey Research," *Software Engineering Notes*, vol. 26 (No. 6), 27 (No. 1, 3, and 5), 28 (No. 2), no., 2001-2003.

[19] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming," European Software Control and Metrics (ESCOM), London, England, 2001.

[20] J. T. Nosek, "The Case for Collaborative Programming," in *Communications of the ACM*, vol. March 1998, March 1998, pp. 105-108.
[21] D. Palmieri, "Knowledge Management through Pair Programming Masters Thesis," in *Computer Science*. Raleigh, NC: North Carolina State University, 2002.
[22] M. C. Paulk, B. Curtis, and M. Chrisis, "Capability Maturity Model for Software Ver. 1.1," Software Engineering Institute CMU/SEI-93-TR, Feb. 24, 1993.
[23] T. Punter, M. Ciolkowski, B. Freimut, and I. John, "Conducting On-line Surveys in Software Engineering," International Symposium on Empirical Software Engineering, Roman Castles (Rome), Italy, 80-89, 2003.
[24] J. Riggs, *Production Systems Planning, Analysis and Control*, Wiley, 1981.
[25] A. G. Rodrigues and T. M. Williams, "System dynamics in software project management: towards the development of a formal integrated approach," *European Journal on Information Systems*, vol. 6, no. pp. 51-56, 1997.
[26] M. Ruiz, I. Ramos, and M. Toro, "A simplified model of software project dynamics," *Journal of Systems and Software*, vol. 59, no. pp. 299-309, 2001.
[27] A. Shukla, "Pair Programming and the Factors Affecting Brooks' Law Master's Thesis," in *Computer Science*. Raleigh: North Carolina State University, 2002.
[28] G. Srinivasa and P. Ganesan, "Pair Programming: Addressing Key Process Areas of the People-CMM," XP/Agile Universe, Chicago, 2002.
[29] R. D. Stutzke, "A Mathematical Expression of Brooks' Law," Ninth International Forum on COCOMO and Cost Modeling, Los Angeles, CA, 1994.
[30] G. M. Weinberg, *Quality Software Management: System Thinking*, vol. I: Dorset House Publishing, 1992.
[31] L. Williams, R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair-Programming," *IEEE Software*, vol. 17, 2000, pp. 19-25.
[32] L. Williams and R. Kessler, *Pair Programming Illuminated*. Reading, Mass.: Addison Wesley, 2003.
[33] L. A. Williams, "The Collaborative Software Process PhD Dissertation," in *Department of Computer Science*. Salt Lake City, UT: University of Utah, 2000.
[34] L. M. Yarroll and S. Hari, "Extreme Programming Practices Promote Teamwork," Motorola System Engineering Symposium, Schaumburg, IL, 2002.

Appendix A: Survey I[3]

1. For this question, assume the new person already has the necessary skills and experience for the job. The new person must only learn project-specific facts such as facility layout, administrative details (staff names, procedures), the development process (e.g. policies, procedures, etc.) and the development domain, as well as the development and test environment.

How much time (in person days) does it take for a new team member to become assimilated to the project work? They become assimilated once they can be "independently" productive and own their own tasks (albeit not such a complex task) without relying HEAVILY on other team members. With a non-pairing programmer, this means they can work without finding someone for help *often*. With a pairing programmer, they can be a contributing partner for more than just simple syntax/tactical defects. Assimilation begins when the person reports to the project to start work.

Number of workdays to assimilate . . .

- Without pair programming: Please state three estimated values. [Low, Most likely, High]
- With pair programming: Please state three estimated values. [Low, Most likely, High]

2. A new person generally becomes assimilated through mentoring/apprenticeship by a team member (as opposed to any formal/class training). During the assimilation time in question #1, what fraction of their mentor's work time is spent helping the new team member? For non-pair programming, this is the fraction of the day spent with a mentor/experienced team member on average during the assimilation period. For pair programming, this is the fraction of the day "lost" when an experienced person pairs with a new person vs. when they pair with another experienced person (on average).

Percent of the experienced person's regular workweek (40 hours) . . .

- Without pair programming: Please state three estimated values. [Low, Most likely, High]
- With pair programming: Please state three estimated values. [Low, Most likely, High]

Appendix B: Survey II[3]

1. When a team member joins your team, which one of the following is your first choice for pairing with the new team member? Please circle your choice.

- experienced programmer (10 + years)
- experienced programmer (5 + years)
- junior programmer (6 months - 3 years)
- another new recruit (< 6 months)

2. The term pair rotation is used to denote when team members' pair with different team members for varying times throughout the project. Does your team practice pair rotation?

[3] Additional questions on the respondent's job title and experience level were included and used to as selection criteria for inclusion in the analysis.