All I Really Need to Know about Pair Programming I Learned In Kindergarten
(submitted to Communications of the ACM)

Laurie A. Williams
University of Utah
lwilliam@cs.utah.edu

Robert R. Kessler
University of Utah
kessler@cs.utah.edu

Abstract

All I Really Need to Know I Learned in Kindergarten
By Robert Fulghum (Fulghum 1988)

Share everything.

Play fair.

Don't hit people.

Put things back where you found them.

Clean up your own mess.

Don't take things that aren't yours.

Say you're sorry when you hurt somebody.

Wash your hands before you eat.

Flush.

Warm cookies and cold milk are good for you.

Live a balanced life – learn some and think some and draw and paint and sing and dance and play and work every day some.

Take a nap every afternoon.

When you go out into the world, watch out for traffic, hold hands and stick together.

Be aware of wonder.

Introduction

Pair programming is a style of programming in which *two* programmers work side-by-side at *one* computer, continuously collaborating on the same design, algorithm, code or test. As discussed below, use of this practice has been demonstrated to improve productivity and quality of software products. Additionally, based on a survey(Williams 1999) of pair programmers (hereafter referred to as "the pair programming survey"), 100% agreed that they had more confidence in their solution when pair programming than when they program alone. Likewise, 96% agreed that they enjoy their job more than when programming alone.

© 1999 Laurie Williams and Robert Kessler

However, most programmers are long conditioned to performing solitary work and often resist the transition to pair programming. Ultimately, most triumphantly make this transition. This purpose of this paper is to aid programmers in becoming effective pair programmers. The transition and on-going success as a pair programmer often involves practicing everyday civility, as written about in Robert Fulghum's poem above. Each of the poem lines (some adapted with poetic license) will be explored for the inherent lessons related to successful pair programming.

Pair Programming: Evidence of Success

Anecdotal and initial statistical evidence indicates pair programming is highly beneficial. *Extreme Programming* (XP), an emerging software development methodology, attributes great success to the use of "pair programming." XP was developed initially by Smalltalk code developer and consultant Kent Beck with colleagues Ward Cunningham and Ron Jeffries. The evidence of XP's success is highly anecdotal, but so impressive that it has aroused the curiosity of many highly-respected software engineering researchers and consultants. The largest example of its accomplishment is the sizable Chrysler Comprehensive Compensation system launched in May 1997. After finding significant, initial development problems, Beck and Jeffries restarted this development using XP principles. The payroll system pays some 10,000 monthly-paid employees and has 2,000 classes and 30,000 methods, (Anderson 1998), went into production almost on schedule, and is still operational today. Additionally, programmers at Ford Motor Company, spent four unsuccessful years trying to build the Vehicle Cost and Profit System (VCAPS) using a traditional waterfall methodology. The XP developers successfully implemented that system in less than a year using Extreme Programming (Beck 1999).

XP attributes great success to the use of "pair programming." All production code is written with a partner. XP advocates pair programming with such fervor that even prototyping done solo is scrapped and re-written with a partner. One key element is that while working in pairs a continuous code review is performed, noting that it is amazing how many obvious but unnoticed defects become noticed by another

© 1999 Laurie Williams and Robert Kessler

person watching over their shoulder. Results (Beck 1999) demonstrate that the two programmers work together more than twice as fast and think of more than twice as many solutions to a problem as two working alone, while attaining higher defect prevention and defect removal leading to a higher quality product.

Two other studies support the use of pair programming. Larry Constantine, a programmer, consultant, and magazine columnist reports on observing "Dynamic Duos" during a visit to P. J. Plaugher's software company, Whitesmiths, Ltd, providing anecdotal support for collaborative programming. He immediately noticed that at each terminal were two programmers working on the same code. He reports, "Having adopted this approach, they were delivering finished and tested code faster than ever . . . The code that came out the back of the two programmer terminals was nearly 100% bug free . . . it was better code, tighter and more efficient, having benefited from the thinking of two bright minds and the steady dialogue between two trusted terminal-mates . . . Two programmers in tandem is not redundancy; it's a direct route to greater efficiency and better quality."(Constantine 1995)

An experiment by Temple University Professor Nosek studied 15 full-time, experienced programmers working for 45 minutes on a challenging problem, important to their organization, in their own environment, and with their own equipment. Five worked individually, ten worked collaboratively in five pairs. Conditions and materials used were the same for both the experimental (team) and control (individual) groups. This study provided statistically significant results, using a two-sided t-test. "To the surprise of the managers and participants, all the teams outperformed the individual programmers, enjoyed the problem-solving process more, and had greater confidence in their solutions." The groups completed the task 40% more quickly and effectively by producing better algorithms and code in less time. The majority of the programmers were skeptical of the value of collaboration in working on the same problem and thought it would not be an enjoyable process. However, results show collaboration improved both their performance and their enjoyment of the problem solving process (Nosek 1998).

© 1999 Laurie Williams and Robert Kessler

The respondents of the pair programming survey gave overwhelming support for the technique. Says one: “I strongly feel pair programming is the primary reason our team has been successful. It has given us a very high level of code quality (almost to the point of zero defects). The only code we have ever had errors in was code that wasn’t pair programmed . . . we should really question a situation where it isn’t utilized.” Given these successes, let's review some principles of pair programming in the context of Fulghum’s poem.

Share everything.

In pair programming, *two* programmers are assigned to jointly produce *one* artifact (design, algorithm, code, etc.). The two programmers are like a coherent, intelligent organism working with one mind, responsible for every aspect of this artifact. One person is typing or writing, the other is continually reviewing the work. But, both are equal participants in the process. It is not acceptable to say or think things such as, “You made an error in your design.” or “That defect was from your part.” Instead, “We screwed up the design.” or, better yet, “We just got through test with no defects!” Both partners own everything.

Play fair.

With pair programming, one person “drives” (has control of the keyboard or is recording design ideas) while the other is continuously reviewing the work. Even when one programmer is significantly more experienced than the other, it is important to take turns “driving,” lest the observer become disjoint, feel out of the loop or unimportant.

The person not driving is not a passive observer, instead is always active and engaged. “Just watching someone program is about as interesting as watching grass die in a desert (Beck to be published).” In the pair programming survey, approximately 90% stated that the main role of the person not typing was to perform continuous analysis, design and code reviews. “When one partner is busy typing, the other is

© 1999 Laurie Williams and Robert Kessler

thinking at a more strategic level – where is this line of development going? Will it run into a dead end? Is there a better overall strategy? (Beck to be published)."

Don't hit your partner.

But, make sure your partner stays focused and on-task. Doubtlessly, a benefit of working in pairs is that each is far less likely to "waste time" reading e-mail, surfing the web, or zoning out the window – because their partner is awaiting continuous contribution and input. Additionally, each is expecting the other to follow the prescribed development practices. "With your partner watching, though, chances are that even if you feel like blowing off one of these practices, your partner won't . . . the chances of ignoring your commitment to the rest of the team is much smaller in pairs then it is when you are working alone (Beck to be published)."

Summarized in the pair programming survey, "It takes more effort because the pace is forced by the other person all the time; neither person feels they can slack off." As each keeps their partner focused and on-task, tremendous productivity gains and quality improvements are realized.

Put things back where they belong.

The mind is a tricky thing. If you think about something enough, the brain will consider it a truth. If you tell yourself something negative, such as "I'm a terrible programmer," soon your brain will believe you. However, anyone can control this negative self-talk by putting these thoughts where they belong, in the trash can, every time they start to creep into their brain. The surveyed pair programmers indicated that it was very difficult to work with someone who had a great insecurity or anxiety about their programming skills. They tend to have an "If I work with you, you might find out I've never coded with exceptions" defensiveness about them. Programmers with such insecurity should view pair programming as a means to improve their skill by constantly watching and obtaining feedback from another.

© 1999 Laurie Williams and Robert Kessler

A survey respondent reflected, “The best thing about pair programming for me is the continuous discussion gave me training in formulating the thoughts I have about design and programming, thereby helping me reflect over them, which has made me a better designer/programmer.” Indeed, two researchers surveyed 750 working programmers on coordination techniques in software development (Kraut 1995). The communication technique with both the highest use and the highest value was “discussion with peers.” “The standard response when one confronts a problem that cannot be solved alone is to go to a colleague close by (Kraut 1995).” When pair programming, the “colleague close by” is continuously available. Together the pair can solve problems they couldn’t solve alone and can help improve each other’s skills.

Also, negative thoughts such as “I’m an awesome programmer, and I’m paired up with a total loser” should also find their place in the trash can, lest the collaborative relationship be destroyed. None of us, no matter how skilled, is infallible and above the input of another. John von Neumann, the great mathematician and creator of the von Neumann computer architecture, recognized his own inadequacies and continuously asked others to review his work. “And indeed, there can be no doubt of von Neumann's genius. His very ability to realize his human limitation put him head and shoulders above the average programmer today . . . Average people can be trained to accept their humanity -- their inability to function like a machine -- and to value it and work with others so as to keep it under the kind of control needed if programming is to be successful (Weinberg 1998).”

Clean up your mess.

Pair programmers cite that it is amazing how many obvious but unnoticed defects become noticed by another person watching over your shoulder. Additionally, these defects can be removed without the natural animosity that might develop in a formal inspection meeting. Established software engineering techniques often stress the importance of defect prevention and efficient defect removal. This “watch over the shoulder” technique, perhaps, epitomizes defect prevention and defect removal efficiency. “Given enough eyeballs, all bugs are shallow (Weinberg 1999).”

© 1999 Laurie Williams and Robert Kessler

Don’t take things too seriously.

“Ego-less programming,” an idea surfaced by Gerald Weinberg in *The Psychology of Computer Programming* (recently re-reported in Weinberg 1998) a quarter of a century ago, is essential for effective pair programming. According to the pair programming survey, excess ego can manifest itself in two ways, both damaging the collaborative relationship. First, having a “my way or the highway” attitude can prevent the programmer from considering others ideas. Secondly, excess ego can cause a programmer to be defensive when receiving criticism or to view this criticism as mistrust.

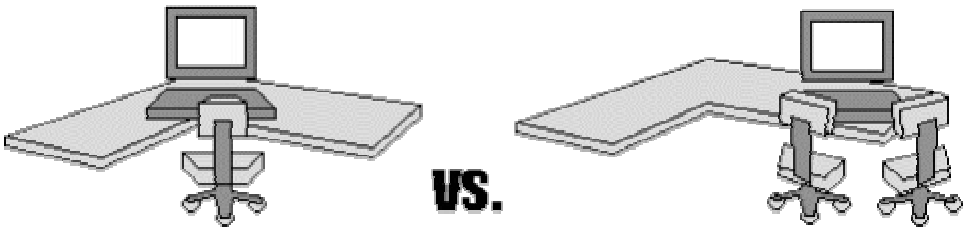
In *The Psychology of Computer Programming (Weinberg 1998)*, a true scenario about a programmer seeking review of the code he produced is discussed. On this particular “bad programming” day, this individual laughed at himself because his reviewer found seventeen bugs in thirteen statements. However, after fixing these defects, this code performed flawlessly during test and in production. How different this outcome might have been had this programmer been too proud to accept the input of others or had viewed this input as an indication of his inadequacies. Having another to continuously and objectively review design and coding is a very beneficial aspect of pair programming. “The human eye has an almost infinite capacity for not seeing what it does not want to see . . . Programmers, if left to their own devices, will ignore the most glaring errors in their output -- errors that anyone else can see in an instant (Weinberg 1998)."

Conversely, a person who always agrees with their partner lest create tension also minimizes the benefits of collaborative work. For favorable idea exchange, there should be some healthy disagreement/debate. Notably, there is a fine balance between displaying too much and too little ego. Effective pair programmers hone this balance during an initial adjustment period. Ward Cunningham, one of the XP founders and experienced pair-programmer, reports that this initial adjustment period can take hours or days, depending on the individuals, nature of work and their past experience with pair-programming.

© 1999 Laurie Williams and Robert Kessler

Say you're sorry when you hurt somebody while moving furniture.

In the pair programming survey, 96% of the programmers agreed that appropriate workspace layout was critical to their success. The programmers must be able to sit side-by-side and program, simultaneously viewing the computer screen and sharing the keyboard and mouse. In the diagram below (from (Beck 1999)), layouts to the right are preferable to layouts on the left. Extreme programmers have a "slide the keyboard/don't move the chairs" rule.



Effective communication, both within a collaborative pair and with other collaborative pairs, is paramount. Without much effort, programmers need to see each other, ask each other questions and make decisions on things such as integration issues, lest these questions/issues are often not discussed. Programmers also benefit from "accidentally" overhearing other conversations to which they can have vital contributions. Separate offices and cubicles can inhibit this necessary exchange. "If any one thing proves that psychological research has been ignored by working managers, it's the continuing use of half partitions to divide workspace into cubicles. … Like many kings, some managers use divide-and-conquer tactics to rule their subjects, but programmers need contact with other programmers. (Weinberg 1998)"

Pair programmers take aggressive action on improving their physical environment, by taking matters into their own hands (armed with screwdrivers).

Wash your hands of skepticism before you start.

© 1999 Laurie Williams and Robert Kessler

Many programmers venture into their first pair programming assignment skeptical of the value of collaboration in programming, not expecting to benefit from or to enjoy the experience. Two skeptical programmers joined together in a team, could certainly carry out this self-fulfilling prophecy. In the pair programming survey, 91% agreed that “partner buy-in” was critical to pair programming success.

Pair programming relationships can be informally started by one programmer asking another to have a seat and give them some help – and carry on from there. Once the relationship has been created, one could say, “That went well. I have some extra time now. Is there anything this afternoon that I can help you with?" Experience has shown that having just one programmer, very positive and/or experienced in pair programming, can lead the pair to victoriously become one jelled collaborative team.

Tom DeMarco shares his inspiring view on this type of union. “A jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts. The production of such a team is greater than that of the same people working in unjelled form. Just as important, the enjoyment that people derive from their work is greater than what you'd expect given the nature of the work itself. In some cases, jelled teams working on assignments that others would declare downright dull have a simply marvelous time. … Once a team begins to jell, the probability of success goes up dramatically. The team can become almost unstoppable, a jaggernaut for success (DeMarco 1977).”

Advice to an up-and-coming pair programmer: Wash your hands of any skepticism, develop an expectation of success, and greet your collaborative partner by saying, “Jell me!” This is an unprecedented opportunity for the two to excel as one.

Flush.

Inevitably, the pair programmers will work on something independently. Of the programmers that were surveyed, over half said that when they rejoined with their partner, they reviewed this independent work and then incorporated it into the project. Alternately, Extreme Programmers flush and rewrite

© 1999 Laurie Williams and Robert Kessler

independent work. In their XP experience, the majority of the defects they found could be traced back to a time when a programmer worked independently. In fact, in their Chrysler Comprehensive Compensation project during the last five months before first production, the only defects that made it through unit and functional testing were written by someone programming alone. In re-writing, the author must undergo the customary continuous review of the work, which identifies additional defects.

The decision to flush or to review work done independently can be made by a pair of programmers, or the choice may be encouraged, as it is with Extreme Programming. However, it is important to note that none of the programmers surveyed incorporated work done independently without reviewing it.

Warm cookies and cold milk are good for you.

Because pair programmers do keep each other continuously focused and on-task, it can be a very intense and mentally exhausting. Periodically, taking a break is important for maintaining the stamina for another round of productive pair programming. During the break, it is best to disconnect from the task at hand and approach it with a freshness when restarting. Suggested activities: checking email, making a phone call, surfing the web, eating warm cookies and drinking cold milk.

Live a balanced life – learn some and think some and draw and paint and sing and dance and play and work every day some.

Communicating with others on a regular basis is key for leading a balanced life. “If asked, most programmers would probably say they preferred to work alone in a place where they wouldn't be disturbed by other people (Weinberg 1998).” But, informal discussions with other programmers – the one you are paired with or any other – allow for effective idea exchange and efficient transfer of information. For example, *The Psychology of Computer Programming* (Weinberg 1998) discusses a large university computing center. A common space with a collection of vending machines was in the back of the room. Some serious-minded students complained about the noise in this common space, and the vending

© 1999 Laurie Williams and Robert Kessler

machines were moved out. After the vending machines were removed and signs urging quiet had been posted, a different complaint abounded – not enough computer consultants! Suddenly, the lines for the computer consultant wound around the room. The cause of the change: the informal chat around the vending machines had consisted of idea exchange and information transfer between the mass of programmers. Now, all this discussion had to be done with the relatively few consultants. (Sadly, the vending machines were never moved back in.)

Take a break from working together every afternoon.

It's certainly not necessary to work separately every afternoon. But, according to 50% of the surveyed programmers, it is acceptable to work alone 10-50% of the time. Many prefer to do experimental prototyping, tough, deep-concentration problems, and logical thinking alone. Most agree that simple, well-defined, rote coding is more efficiently done by a solitary programmer and then reviewed with a partner.

When you go out into the world, watch out for traffic, hold hands and stick together.

With pair programming, the two programmers become one. There should be no competition between the two; both must work for a singular purpose, as if the artifact was produced by a singular good mind. Blame for problems or defects should never be placed on either partner. The pair needs to trust each other's judgement and each other's loyalty to the team.

Be aware of the power of two brains.

Human beings can only remember and learn a bounded amount. Therefore, they must consult with others to increase this bounded amount. When two are working together, each has their own set of knowledge and skills. A large subset of this knowledge and these skills will be common between the two, allowing them to interact effectively. However, the unique skills of each individual will allow them to

engage in interactions which pool their resources to accomplish their tasks. “Collaborative people are those who identify a possibility and recognize that their own view, perspective, or talent is not enough to make it a reality. Collaborative people see others not as creatures who force them to compromise, but a colleagues who can help them *amplify their talents and skills* (Hargrove 1998).”

Experiences show that, together, a pair will come up with more than twice as many possible solutions than the two would have working alone. They will then proceed to more quickly narrow in on the “best” solution and will implement it more quickly and with better quality. A survey respondent reflects, “It is a powerful technique as there are two brains concentrating on the same problem all the time. It forces one to concentrate fully on the problem at hand.”

Summary

Anecdotal and initial statistical evidence indicates that pair programming is a powerful technique for productively generating high quality software products. The pair works and shares ideas together to tackle the complexities of software development. They continuously perform inspections on each other’s artifacts leading to the earliest, most efficient form of defect removal possible. In addition, they keep each other intently focused on the task at hand.

Programmers, however, have generally been conditioned to performing solitary work, rooted at an educational system of individual evaluation, perhaps, to the exclusion of learning. Making the transition to pair programming involves breaking down some personal barriers beginning with the understanding that *talking is not cheating*. First, the programmers must understand that the benefits of intercommunication outweigh their common (perhaps innate) preference for working alone and undisturbed. Secondly, they must confidently share their work, accepting instruction and suggestions for improvement in order to improve their own skills and the product at hand. They must display humility in understanding that they are not infallible and that their partner has the ability to make improvements in what they do. Lastly, a

© 1999 Laurie Williams and Robert Kessler

pair programmer must accept ownership of their partner's work and, therefore, be willing to constructively express criticism and suggested improvements.

The transition to pair programming takes the conditioned solitary programmer out of their "comfort zone." However, the potential for achieving results impossible by a single programmer makes this a journey to greatness.

Bibliography

Anderson, A., Beattie, Ralph, Beck, Kent et al. (1998). Chrysler Goes to "Extremes". Distributed Computing. **October 1998:** 24-28.

Beck, K., Cunningham, Ward (1999). Extreme Programming Roadmap, http://c2.com/cgi/wiki?ExtremeProgramming. **1999**.

Beck, K. (to be published). Embrace Change: Extreme Programming Explained, Addison-Wesley.

Constantine, L. L. (1995). Constantine on Peopleware. Englewood Cliffs, NJ, Yourdon Press.

DeMarco, T., Lister, Timothy (1977). Peopleware. New York, Dorset House Publishers.

Fulghum, R. (1988). All I Really Need to Know I Learned in Kindergarten. New York, Villard Books.

Hargrove, R. (1998). Mastering the Art of Creative Collaboration, McGraw-Hill.

Kraut, R. E., Streeter, Lynn A. (1995). "Coordination in Software Development." Communications of the ACM **March 1995**(March 1995): 69-81.

Nosek, J. T. (1998). The Case for Collaborative Programming. Communications of the ACM. **March 1998:** 105-108.

Weinberg, G. M. (1998). The Psychology of Computer Programming Silver Anniversary Edition. New York, Dorset House Publishing.

Weinberg, G. M. (1999). Egoless Programming. IEEE Software. **January/February:** 118-120.

Williams, L. (1999). Pair Programming Questionnaire, http:\\limes.cs.utah.edu/questionnaire/questionnaire.htm. **1999**.

© 1999 Laurie Williams and Robert Kessler