Laurie A. Williams and
Robert R. Kessler

All I Really to know about P Programmi I Learned Kindergar

When it comes to programming practices, studies show two heads are almost always better than one.

NEED PAIR [ING IN TEN

Pair programming is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test. This method has been demonstrated to improve productivity and the quality of software products. Moreover, a recent survey (hereafter referred to as "the pair programming survey") found that programmers were universally more confident in their solutions when programming in pairs as opposed to working alone. Likewise, 96% agreed they enjoy their jobs more when pair programming [12].

However, most programmers are long conditioned to working alone and often resist the transition to pair programming. Ultimately, most make this transition with great success. The goal of this article is to help programmers become effective pair programmers. The transition to and on-going success as a pair programmer often involves practicing everyday civility, as illustrated in an essay by Robert Fulghum (see box). Here, we take each line from the essay (with occasional poetic license) to explore the inherent lessons related to successful pair programming.

Anecdotal and initial statistical evidence indicates pair programming is highly beneficial. In extreme programming (XP)—an emerging software development methodology—all production code is written with a partner. XP was developed initially by Smalltalk code developer and consultant Kent Beck with colleagues Ward Cunningham and Ron Jeffries. XP's requirements gathering, resource allocation, and design practices are a radical departure from most accepted methodologies. Customer requirements are written as fairly informal "User Story" cards where a

rough effort estimate is assigned to the cards. The cards are then designated for a programming pair, and coding begins. With no formal design procedures or discussions on overall system planning or architecture, the pair determines which code in the code base needs to be added or changed. This practice requires the use of *collective code ownership* whereby any programming pair can modify or add to any code in the code base, regardless of the original programmer. Extensive unit testing is continually performed on this ever-enlarging code base.

The evidence of XP's success is highly anecdotal, but so impressive 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 employees each month and has 2,000 classes and 30,000 methods [1]. It went into production almost on schedule, and is still operational today.

MOST PROGRAMMERS ARE LONG CONDITIONED TO WORKING ALONE AND OFTEN RESIST THE TRANSITION TO PAIR PROGRAMMING. ULTIMATELY, HOWEVER, MOST MAKE THIS TRANSITION WITH GREAT SUCCESS.

XP attributes great success to the use of pair programming by all programmers—experts and novices alike. XP advocates pair programming with such fervor that even prototyping done solo is scrapped and rewritten with a partner. One key element is that a continuous code review is performed while working in pairs. It is amazing to see how many obvious, yet unnoticed, defects are recognized when another person is watching over a shoulder. According to [11], the results demonstrate that two programmers working together are 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.

In addition, two other studies support the use of pair programming. Larry Constantine, a noted programmer and consultant, reported on some "dynamic duos" during a visit to P.J. Plaugher's software company, Whitesmiths, Ltd., providing anecdotal support for collaborative programming. He immediately noticed a room full of paired programmers working on the same code at one computer. "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, he contends." [3].

An experiment by John Nosek at Temple University 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, 10 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," Nosek explains.

Moreover, 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 [8].

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 … utilized."

Examination of why pair programming works with such success reveals that a number of elementary

principles come into play. These principles can be discussed in the context of Fulghum's essay:

Share everything.

In pair programming, two programmers are assigned to jointly produce one artifact (design, algorithm, code, among others). 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. 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 our 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 disjointed, feel out of the loop, or unimportant.

The person not driving should not be a passive observer, but instead should always be active and engaged. "Just watching someone program is about as interesting as watching grass die in a desert" [2]. In the pair programming survey, approximately 90% stated 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 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?"

Don't hit people.

Make sure he or she stay focused and on-task (nonviolently, of course). Undoubtedly, a benefit of working in pairs is that each person is far less likely to waste time reading email, Web surfing, or staring out the window because their partner is awaiting continuous contribution and input. "Two people working together in a pair treat their shared time as more valuable. They tend to cut phone calls short; they don't waste each other's time" [10].

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" [2].

As 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 his or her partner focused and on-task, tremendous productivity gains and quality improvements are realized.

Put things (especially negative thoughts) *back where they belong.*

The mind is a tricky thing. If you think about something long 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—out of mind—every time they start to creep in. The surveyed pair programmers indicated it was very difficult to work with someone who had insecurity or anxiety about their programming skills. They tend to have a 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.

A survey respondent reflected, "The best thing about pair programming for me is the continuous discussion that gave me training in formulating the thoughts I have about design and programming. It helps me reflect over these thoughts, which has made me a better designer/programmer." Indeed, two researchers surveyed 750 working programmers on coordination techniques in software development [7]. 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." 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 be rejected, 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" [9].

Clean up your mess.

Pair programmers have the advantage of the presence of a partner to help them clean up. Many have related that many obvious, but undetected, defects were noticed by another person watching over their 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. Perhaps this "over the shoulder" technique epitomizes defect prevention and defect removal efficiency.

NONE OF US, NO MATTER HOW SKILLED, IS INFALLIBLE AND ABOVE THE INPUT OF ANOTHER.

Don't take things too seriously.

"Ego-less programming," an idea that surfaced 25 years ago by Gerald Weinberg in *The Psychology of Computer Programming*, 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 other ideas. Secondly, excess ego can cause a programmer to be defensive when receiving criticism or to view this criticism as mistrust.

A true scenario about a programmer seeking review of the code he produced is discussed in [9]. On this particular bad programming day, an individual egolessly laughed because his reviewer found 17 bugs in 13 statements. After fixing these defects, however, the code performed flawlessly during testing and in production. How different this outcome might have been had the programmer been too proud to accept the input of others or had viewed this input as an indication of his inadequacies. Having another review design and coding continuously and objectively is an extremely 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" [9].

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 this initial adjustment period can take hours or days, depending on the individuals, nature of work, and their past experience with pair programming.

Say you're sorry when you hurt somebody.

In the pair programming survey, 96% of the programmers agreed that appropriate workspace layout was critical to their success. Pair programmers take aggressive action on improving their physical environment, by taking matters into their own hands (armed with screwdrivers). The programmers must be able to sit side-by-side and program, simultaneously viewing the computer screen and sharing the keyboard and mouse. 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 not discussed adequately. 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" [9].

Wash your hands before you start.

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 pro-

grammers 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 established informally 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 become one victoriously jelled collaborative team.

Tom DeMarco shares his inspiring view on this type of union in [4]. "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 juggernaut for success."

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 surveyed, over half said they reviewed work done independently when they rejoined with their partner, and incorporated it into the project. Alternately, extreme programmers flush and rewrite independent work. In their XP experience, the majority of the defects could be traced back to a time when a programmer worked independently. In fact, during the five months prior to first production from the Chrysler Comprehensive Compensation project, the only defects that made it through unit and functional testing were written by someone programming alone. In rewriting, 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 XP. However, it is important to note none of the programmers surveyed incorporated work done independently without reviewing it.

Warm cookies and cold milk are good for you.

Because pair programmers must keep each other continuously focused and on-task, it can be a very intense and mentally exhausting experience. Taking a break periodically 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 refreshed when restarting. Suggested activities: checking email, making phone calls, 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" [9]. 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, Weinberg [9] discusses a large university computing center, in this case a common space with a collection of vending machines in the back of the room. Some of the more serious students complained about the noise in this common space, and the machines were moved out. Soon after the removal of the machines, a different complaint echoed the walls: Not enough computer consultants! Suddenly, the lines for the computer consultant wound around the room. The cause of the change was the fact that informal chat around the vending machines offered idea exchanges and information transfers 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 nap (or 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 wonder* (and the power of two brains working together).**

Human beings can only remember and learn a bounded amount. Therefore, they must consult with others to increase this bounty. 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 that 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 as colleagues who can help them amplify their talents and skills" [6].

Experiences show that a pair will come up with more than twice as many possible solutions as two individuals 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."

Final Thoughts

Both anecdotal and initial statistical evidence indicate that pair programming is a powerful technique for 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 working alone. Making the transition to pair programming involves breaking down some personal barriers. First, the programmers must understand the benefits of intercommunication outweigh their common (perhaps innate) preferences 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 they are not infallible and their partner has the ability to make improvements in what they do. Lastly, a pair programmer must accept ownership of his or her partner's work and, therefore, be willing to constructively express criticism and suggested improvements.

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

References

1. Anderson, A., Beattie, R., Beck, K. et al. Chrysler goes to "Extremes." *Distrib. Comput.* (Oct. 1998), 24–28.
2. Beck, K. *Extreme Programming Explained: Embrace Change.* 1999. Addison-Wesley, Reading, PA.
3. Constantine, L. L. *Constantine on Peopleware.* Yourdon Press, Englewood Cliffs, NJ. 1995.
4. DeMarco, T., Lister, T. Peopleware. Dorset House, New York, NY. 1977.
5. Fulghum, R. *All I Really Need to Know I Learned in Kindergarten.* 1988. Villard Books, New York, NY.
6. Hargrove, R. *Mastering the Art of Creative Collaboration*. McGraw-Hill, New York, NY. 1988.
7. Kraut, R. E., Streeter, L.A. Coordination in software development. *Commun. ACM 38,* 3 (Mar. 1995), 69–81.
8. Nosek, J. T. The case for collaborative programming. *Commun. ACM 41*, 3 (Mar. 1998), 105–108.
9. Weinberg, G. M. *The Psychology of Computer Programming Silver Anniversary Edition.* Dorset House, New York, NY. 1998.
10. Wiki. Pair Programming Facilities. Portland Pattern Repository. Mar. 16, 1999; c2.com/cgi/wiki?PairProgrammingFacilities.
11. Wiki. Programming In Pairs. Portland Pattern Repository. June 29, 1999; c2.com/cgi/wiki?ProgrammingInPairs.
12. Williams, L. Pair Programming Questionnaire. 1999; limes.cs.utah.edu/questionnaire/questionnaire.htm.

Laurie A. Williams (lwilliam@cs.utah.edu) is a Spring 2000 computer science Ph.D. graduate and instructor at the University of Utah, Salt Lake City, UT.

Robert R. Kessler (kessler@cs.utah.edu) is a professor and chair of the Department of Computer Science at the University of Utah, Salt Lake City. He is the founder of the Center for Software Science, a state of Utah Center of Excellence.