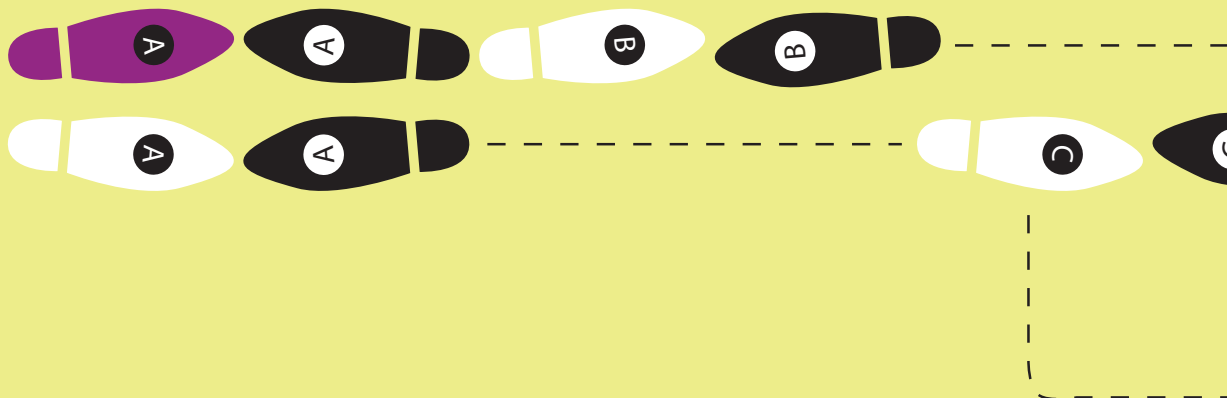
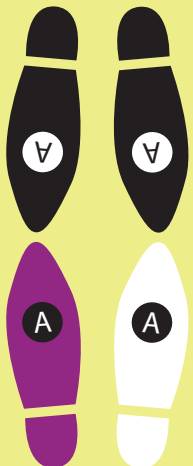


IT TAKES

TWO TO

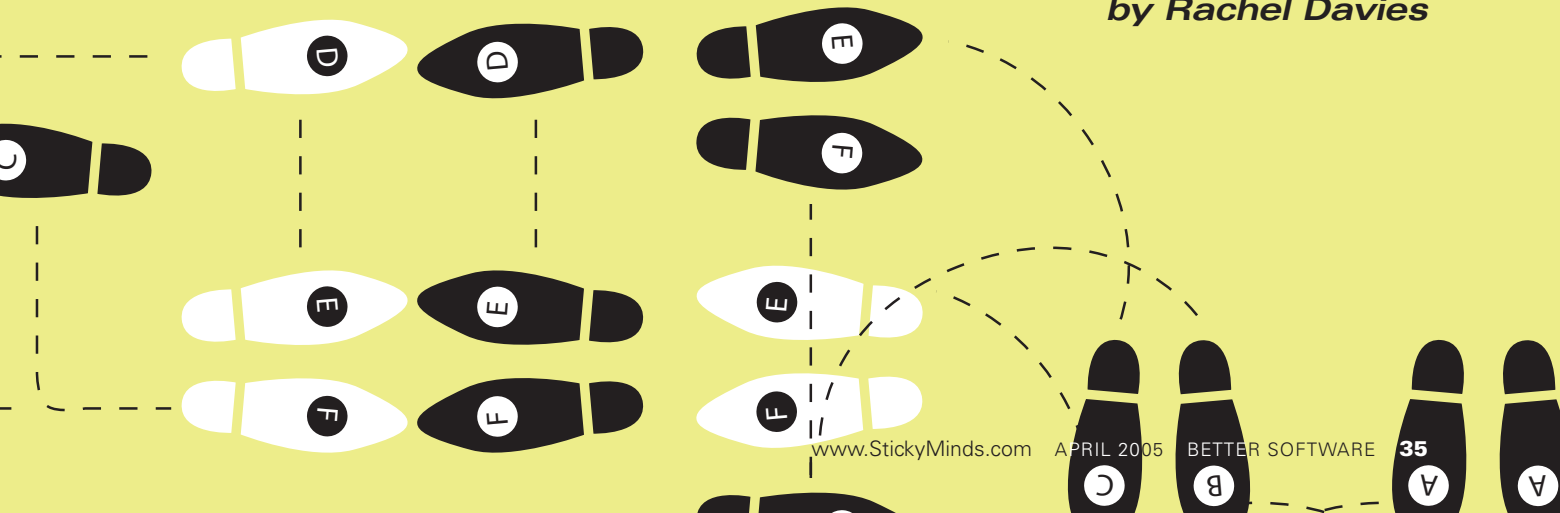




**What every software manager should know about pair programming and how to implement it without missing a step.**

# TANGO

*by Rachel Davies*



Even the best programmer will make mistakes—defects are inevitable. Your strategy for deploying programmers in the most productive way needs to recognize this. Research has shown that programming in pairs can significantly improve code quality with a relatively small trade-off in development time. (See the StickyNotes for reference.) While this sounds counter-intuitive (surely, solo programmers working in parallel will be able to implement more features in the same time), productivity cannot be sustained purely by maximizing output without attention to quality control. This article explains the ins and outs of pair programming and what factors you need to consider before rolling it out in your organization.

### Pairing Up

Pair programming is not one person passively watching the other typing. When engaged in pair programming, each programmer plays an active role in determining the design, implementation, and tests. The pair analyzes strands of

the problem while passing the keyboard back and forth between them, writing code and tests as they go.

Pair programming is a primary practice of Extreme Programming (XP), one of the increasingly popular Agile software development methods. (See the StickyNotes for more on Extreme Programming.) XP is built on the idea that a focus on quality will decrease project risk. An XP team will typically write all production code in pairs. But you don't have to implement XP to benefit from pair programming. Even programmers on non-XP teams naturally gravitate toward working in pairs when faced with difficult problems. It is possible to integrate the practice of pair programming within other approaches, Agile or traditional.

How does a typical pair programming episode work? The pair sits at a single workstation. The person at the keyboard is the driver and directly implements the solution; her partner is the navigator and thinks at a strategic level about the next steps and looks for pitfalls. Pairs switch fluidly between these roles. Each episode

typically lasts a couple of hours (including natural breaks) and ends with code being integrated and checked into version control.

In addition to switching roles during a pair programming episode, programmers typically change partners more than once a day. Exposing code to other programmers improves code quality in at least two ways. First, each programmer who reads the code can illuminate previously unrecognized blind spots that could cause defects. Second, peer pressure can improve pride in work and attention to design. The success of the open source movement hinges on the effect of “many eyes,” and pair programming takes that effect to its logical conclusion.

### Don't Miss a Beat

The scientific management principles that might be used to optimize a production line are not directly applicable to software development because programming is not a manufacturing task. Each development task needs to satisfy some unique feature requirement that must be woven into the complex fabric of a larger system without disrupting the existing behavior. Programming is knowledge work, and software development needs to be managed with knowledge transfer in mind.

It has long been established that code inspections are cost-effective ways to reduce defect rates. (See the StickyNotes for reference.) However, industry adoption of formal inspections remains low. Pair programming provides an alternative way to implement design and code reviews. The peer review that takes place during pair programming prevents poorly designed code from ever being checked in. Since the pair talks through several design approaches, the individual programmer is less likely to form an emotional attachment to her design, an attachment that might impede future rework.

Fixing defects found in the QA phase—or even worse, after deployment—is costly. Leaving code defects to be picked up downstream eventually slows down the software development effort as programmers try to build new features on over-complicated and possibly malfunctioning

## The Evidence

You may be convinced by the claims listed in this article, but what empirical data is there to support them?

1. A 1975 study of “two-person programming teams” reported a 127% gain in productivity and an error rate that was three orders of magnitude less than normal for the organization under study. (See the StickyNotes for a link to the study.)
2. In 1992, Larry Constantine wrote about a team, set up by author P.J. Plauger, which developed code that was nearly 100% bug free. (See the StickyNotes for a link.)
3. Dr. Laurie Williams, author of *Pair Programming Illuminated*, has carried out the main research on this topic. Much of her research can be found on the pair programming website [www.pairprogramming.com](http://www.pairprogramming.com). Williams found in her studies that in exchange for a 15% increase in development time, pair programming improved design quality, reduced defects by an average of 15%, and was reported as more enjoyable by programmers at statistically significant levels. A less well-known finding was that pair programmers generated more concise output—implementing the same functionality in fewer lines of code.

The conditions under which academic research is carried out are unlikely to be an exact match for your software development organization, so perhaps the best way to evaluate pair programming is to try this practice on a small, low-risk project.

code. Pair programming can prevent this slowdown by keeping code clean and well organized at all times.

### Company Benefits

A pair programming team creates a collaborative environment that supports knowledge sharing. As they pair with one another, programmers pick up design tips and new language features. Code inspections also support knowledge sharing but, because they typically happen weeks apart, they have a built-in latency. Discussing a technique in a formal meeting is not the same as actually trying it out. When you are pair programming, you learn by implementing the new concept on real code with your pair on hand to give guidance. Information rapidly diffuses through a pair programming team as pairs switch and learn from each other. Ken Auer, one of the early adopters of XP, calls this knowledge transfer effect the “pair-vine.” The advantage to the company is that each member of the team becomes more versatile, better adapted to absorb any curve ball thrown.

An additional benefit of pair programming is that there are fewer bottlenecks caused by holidays or absences because all code is developed by more than one programmer. You stop having to plan around key people on the critical path. Jim Coplien coined the term “truck number” for the total number of people on a project that, if one of them got hit by a truck, the project would be in trouble. When you rely too heavily on experts, you are betting on their continued availability for work. If these experts get to know how much you rely on them, they can become prima donnas—getting picky about what they will work on and expecting to be well remunerated in return. Pair programming spreads expertise across a team and is one way to reduce the project’s truck number.

When programmers work alone, it can be difficult to tell whether they are actually working on the planned tasks. Here are some problems with the traditional solo-programming model:

- A solo programmer may get distracted and start “gold-plating” code—inventing

## No Silver Bullet

After reading this article it might appear that pair programming is going to have a positive effect on every software development team. But as the mighty Frederick Brooks said, “there is no silver bullet.” (See the StickyNotes for reference.) Here are some scenarios in which pair programming is unlikely to deliver the full set of benefits:

### Flextime and Telecommuting

For pair programming to get off the ground successfully, you need to have programmers available for work in the office for core hours. This may be an issue if your company explicitly offers employees the option to work from home and programmers on the team do not wish to give up this benefit. Contractors may be accustomed to putting in their hours as they please.

### Distributed Teams

It may not make sense to attempt pair programming if the team is split across different company sites. To implement distributed pair programming, you may need to invest in more specialized tools to support remote desktop sharing and voice communication. (For more on this topic, see the article “Remote Control” in the April 2004 issue of *Better Software*.) An alternative is to pair program only within the subteams at each site, then implement code reviews of selected work products across boundaries.

### Diverse and Small Teams

A classic situation in which pair programming does not work well is when you have heterogeneous system architecture and the team made up of a small number of specialists, each proficient in only one technology used in the system. An example of this situation is the specialized database programmer. It may not make sense for an Oracle programmer to pair with a Java programmer because the skills required to work proficiently in their respective technologies are not easy to absorb via pair programming alone, and separate training support may be required.

Very small teams (one to three programmers) may find that they do not benefit from formal pair programming as they already have a high level of interaction and might find working with the same person too claustrophobic.

things that were not asked for because the inventions are more interesting than the task at hand.

- A lone programmer may get stuck. The temptation is to surf the Web or stop working on that task and start another rather than lose face by asking for help. This meandering can lead to a state where several tasks are paused at “90 percent done.”

- A lazy programmer may skimp on parts of the development process with her short cuts going undetected. For example, she may check in code without unit tests or include code that does not build, which may have the effect of slowing down other programmers on the project.

When programmers write code in pairs, they keep each other on task and on process.

### Programmer Benefits

Many of the company benefits also turn out to be programmer benefits. Working with a partner means there is always someone there to help you get unstuck. By staying focused on the task at hand, you are likely to have working code at the end of the day, so you can go home with a warm glow of achievement. Curbing programmers with bad habits—those who create work for the rest of the team by checking in broken code—can be a relief to those programmers who were cleaning up in their wake.

Another benefit to programmers is

that it releases them from being pigeon-holed, held back from developing their skills because they are too precious to release from specialist work that no one else knows how to do.

But not all programmers embrace the idea of working in pairs. Fearing that any gaps in coding skills will be quickly exposed, programmers may be reluctant to try pair programming. My experience is that once programmers try pair programming this concern quickly evaporates. Because you are learning from your programming partners, gaps in coding skills are quickly filled. When pairing, a programmer has to keep on her toes, and most developers enjoy this challenge.

### Implementation

First, you will need to address the working environment. To pair program, the two programmers need to be able to program at one workstation in comfort. Each workstation must have a large

screen area so that both partners can read the code without having to lean in close. Dual screens are becoming a popular way to implement this. Additionally, a cordless keyboard and mouse may be easier to pass between partners.

Many offices are equipped with desks designed for a single programmer to sit in an inset. It is impossible to pair program at these desks without crowding your partner. Instead, you will need either straight-sided tables or, even better, convex curved desks. (See this issue's Sticky Notes for an example of an effective setup.) Some teams install a "bullpen" with a central island of tables set up for pair programming. If you do this, remember that teams still need some space to check personal email and make telephone calls.

### Practice Makes Perfect

It takes time to become skilled at pair programming. Because the technique hinges on improving communication

skills, it is vital that you articulate your ideas and listen to the pair. There are various hands-on training courses that provide an introduction to Agile software development techniques, but it is more common to bring in experienced people to work with your team as player-coaches.

Programmers who are new to pair programming may need to be reminded to take turns at the keyboard. There are some simple techniques that can be used to keep pairs on track. I have heard of a team that uses a chess clock to ensure that each partner takes an equal turn at driving and navigating. Another team uses a kitchen timer to synchronize pair programming episodes and break time.

The team needs to come to some agreement on coding standards. Consistent naming and layout conventions help make the code more readable and thus reduce the time that it takes to get up to speed working on unfamiliar code. Each

## Serving Up Clean Code

What do you get when you cross Test Driven Development with pair programming and add a dash of competitive fun? A technique called Ping-Pong programming—something that software developer and ThoughtWorks consultant Dave Hoover finds to be helpful for structuring the pair programming effort.

### PERSPECTIVE

get situations with long sessions of one person driving the programming and one person watching and helping. Sometimes that's appropriate. But it can also get boring." Ping-Pong programming is a way to keep the keyboard moving back and forth between the two. As Hoover has experienced, this not only facilitates the flow but also tends to draw reluctant collaborators into the process by introducing a game-like element to the mix.

Here's how it works. Test Driven Development is based on small steps—devising a test the intended code should pass, writing the code to pass the test, and so on. For example, picture a pair of developers—Fred and Jo. Fred writes a test and then passes the keyboard to Jo, who then writes the code to pass the test. Jo writes the next

test and passes the keyboard back to Fred, who writes the code to pass *that* test. Back and forth the keyboard slides as Fred and Jo build the functionality, refactoring throughout. Hence the term "Ping-Pong."

But, as Hoover explains, it's not just about employing a more interesting, interactive way to pair program. "The person writing the code tries to pass each test with as little code as possible. This leads the person writing the test to make each test more rigorous and more difficult to pass." The result is simple, testable code—and that makes for good design.

There are other benefits to Ping-Pong programming, too. Hoover finds the active interaction, more so than simple observation, is a great way to pick up on the design experience of the person with whom he pairs. He has also noticed that the element of fun tends to keep people more engaged.

So what do managers need to know about how and when this form of pair programming works best? Hoover suggests explaining the principle to your developers and encouraging them to try it for awhile. "It's a good tool if your developers are in a rut or if you're trying to get pair programming going and want to give them a model to follow rather than just saying 'do it'."

It may not be long before your developers are volleying for cleaner and better code.—PY

*Dave Hoover has joined the ranks of our StickyMinds.com weekly columnists. Catch the first of his columns in June; log on to [www.StickyMinds.com](http://www.StickyMinds.com).*

workstation should have the same set of development tools so that a pair can use any free workstation.

As with any change, it is important to communicate to the team the reasons for adopting pair programming and agree how the approach is to be evaluated. Remember, it is perfectly natural to resist change. You may need to start gradually with programmers pairing for only part of the day. Programmers may be concerned that sharing knowledge will lead to loss of status, so you should be clear that performance reviews will not be based on completion of individual tasks alone but also will take into account contributions to team goals.

### Selecting Partners

If you presently work to project plans that allocate tasks on an individual basis, then you may be intrigued to know how pair programming will affect task allocation and resourcing in your planning tool. Task allocation may remain unchanged on the assumption that each programmer pairs on her own tasks for part of the time and for the remaining time makes herself available to work with other programmers. Remember that task durations will be slightly longer (say 20 percent, based on the empirical studies and taking into consideration that the team is adopting a new development technique). You should make up time later because there will be less time spent fixing bugs found by the QA effort.

At the start, the team lead may prefer to assign partners to ensure each pair has the right experience for the task. Many Agile teams allocate pairs at a stand-up meeting at the start of each day. I worked on a team where we used a simple pair rotation rule: Each day the pair would split, leaving one person on the task for continuity and releasing the other to pair on another task. You may anticipate that when inexperienced programmers pair with expert programmers then the experts will be frustrated by having to slow down for their partner. It turns out that novices can complement experts and quickly learn new skills. Experts benefit from broadening their skills and picking up tips on newer libraries, etc. Be aware that pair

programming should not be used as a substitute for training. To avoid pair cliques forming and to ensure that each team member gets the opportunity to pair with all the others, it may be wise in the first few weeks to track pairing combinations.

### Going Solo

There are times when solo programming is appropriate. When developing a temporary solution or prototype, code quality is not so important, so splitting a pair makes sense when exploring alternative technical solutions or bug-busting where different causes need to be eliminated. There will be days when you have an odd number of programmers. Most programmers find it a little claustrophobic to be locked into pair programming all the time and appreciate some time to work on their own now and then. Just make sure any production code written without a pair is peer reviewed before being checked in.

When recruiting new team members, consider including a pair programming session as part of your interview process. This will allow your existing programmers a good opportunity to assess the candidate and will give the candidate a chance to

try the team's work practices. This should help weed out programmers with poor communications skills and those who prefer to work solo.

To be assured that the practice is delivering results, keep tabs on the reported defect rate—you should start to get fewer critical bugs reported. After the first few weeks, hold a retrospective with the team to explore how the technique is working. **{end}**

*Rachel Davies is an independent Agile coach in the UK, a frequent presenter at industry conferences, and a director of the Agile Alliance. Contact her at Rachel@agilexp.com.*

### Sticky Notes

For more on the following topics go to [www.StickyMinds.com/bettersoftware](http://www.StickyMinds.com/bettersoftware)

- More on Extreme Programming
- More on Inspections
- Link to two-person programming team study
- Link to Constantine paper
- Suggested workstation layout

## Tell Us What You Think

**"I HAVE NEVER RECEIVED AN ISSUE ... WHERE I DID NOT FIND AT LEAST TWO, AND OFTEN AS MANY AS FIVE OR SIX, ARTICLES THAT I STRONGLY ENCOURAGED SOMEONE ELSE TO READ. AND I SAVE EVERY ISSUE FOR REFERENCE.**

—respondent to our Subscriber Survey

**WHAT ABOUT YOU?** Tell us what you like—and don't like—about this or any other issue of *Better Software*. Email your comments to [editors@bettersoftware.com](mailto:editors@bettersoftware.com) or fax them to us at **904-278-4380**.

Comments may be edited for space and readability and published in a future issue of *Better Software*.