Extracted from:

Remote Pairing

Collaborative Tools for Distributed Development

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic Press

Remote Pairing

Collaborative Tools for Distributed Development

Joe Kutner

Edited by Brian P. Hogan

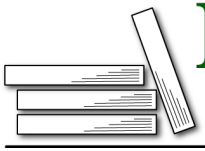
Remote Pairing

Collaborative Tools for Distributed Development

Joe Kutner

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Brian P. Hogan (editor)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-74-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P2.0—January 2014

Pairing at Pivotal Labs

Pivotal Labs is a software firm based in San Francisco.[4] You may be familiar with its flagship product, Pivotal Tracker,[5] but the company does a wide range of other software development—client work, iPhone apps, and more.

Pivotal uses a strict system of pair programming, in which most developers pair with another developer every time they sit down to write code. That's eight hours a day, five days a week. The majority of Pivotal's employees work in one of the company's seven major offices, but a handful of developers work from remote locations. Despite the distance, these anywhere-based developers adhere to the same strict pairing policy.

One of Pivotal's veteran remote developers is Joe Moore, who's become an outspoken advocate for remote pair programming. His blog is an excellent source for pairing-technology news, and he's given a number of public talks on the subject.[6,7] Joe started as an on-site Pivotal employee, but moved out of the office in 2010 so his wife could continue her medical career. Despite his change of location, things at Pivotal remained largely the same.

Joe's morning starts with a stand-up meeting where he and his team decide who will pair with who that day. "We try to swap pairs every day if we can," he says. "If we can't, we try to monitor people who have paired together too many days in a row [because] it's usually an indication that they're stuck on something."

Given Joe's many partners, his stack of software tools varies significantly from day to day. "It depends on the logistics of the client, network speed, security, and VPN," he says. Most of the time, Joe can rely on tmux and Vim, but since he's primarily a web developer he usually needs a screen-sharing tool to supplement his text editor. He prefers either Screenhero or the built-in Mac Screen Sharing app we discussed in Chapter 4, *Collaborating with Shared Screens*, on page ?.[8] These software tools are significant, but they aren't as important to Joe as the human element. "I find that [video] is invaluable," he says. "I want to be able to see people's faces. I want to know if they're confused, if they're laughing, or if they're looking down at their phone." His preference has led to a very elaborate home-office setup. Joe has

4. http://pivotallabs.com/
5. https://www.pivotaltracker.com/
6. http://remotepairprogramming.com/
7. http://remotepairprogramming.com/tagged/video
8. http://screenhero.com/

multiple monitors and a retractable arm that holds an iPad for running his Skype sessions. “It looks like Fort Knox or something,” he muses.

Video is important because it allows Joe to quickly pick up on subtle cues that might otherwise be missed. But sometimes there’s just no replacement for telling your partner what you’re thinking. “I can’t see your hands and you can’t see my hands, so all day long I’m saying things like, ‘Hey, I’m going to grab the mouse’ or, ‘Do you mind if I look at something?’ instead of just doing it,” he explains. “I replace a lot of [visual cues] with verbal cues.” Joe believes that verbalizing everything leads to better pairing etiquette. Many of his on-site pairing partners say that learning to do this has helped improve their colocated pair sessions, too.

Joe has been able to use video and verbal cues to keep his pairing sessions extremely fluid. He’s found that the advantage of having more than one person working on a problem is that you have more ideas, which makes it easier to overcome obstacles. “There’s so much shared experience [that] there’s seldom a time when we’re truly stuck,” he says. “Once you get down to it…it stops being novel and just becomes the way you work.”

Joe loves pairing with his Pivotal team, and he finds that doing eight hours a day works perfectly for him. But he acknowledges that a routine of 100 percent pairing might not be perfect for everyone: “Some people don’t have a good setup at home. Maybe they don’t feel comfortable or there are too many distractions, but some pairing is better than none.” For some teams, the best time to get that little bit of daily remote pairing is when they’re having problems.

Pairing at Big Nerd Ranch

Big Nerd Ranch, an industry leader in iOS development and training,[9] doesn’t have a strict policy of pairing like Pivotal Labs. Instead, developers decide on their own when pairing is appropriate. Jay Hayes, a programmer at Big Nerd Ranch, says this leads to an informal remote-pairing environment. Jay lives in Alabama, 200 miles away from the company headquarters in Atlanta. Most of his team members are based in Atlanta, but none of them are required to make regular trips to the office. Despite the distance, they still manage to pair up on tough problems.

“It doesn’t usually start with ‘Hey, why don’t we pair a bit when you have some downtime,’” says Jay. “It’s more like ‘I’ve been banging my head against the desk, and can anyone help me crush this bug?’” Jay believes this ad hoc

9. http://www.bignerdranch.com

approach helps his team function more like a colocated one. “Our pairing ends up being just like if you were in the office and needed help. Someone might come over and sit with you for a bit—it’s a remote version of that,” he feels. Despite using an “as needed” approach to pairing, Jay does it about once a day—but the sessions don’t often last long: “Maybe thirty minutes on average…and we don’t use any formal methods.”

Jay’s preferred tech stack is tmux and Vim, which he uses for both pairing and working solo. The team uses Google Hangouts to share screens, but they occasionally resort to using Screenhero when they need two-way control, a browser, or an integrated development environment.

Jay is pretty handy with Vim, but that wasn’t always the case. Upon joining Big Nerd Ranch, he knew how to open Vim, exit Vim, move up and down, and nothing else. “[Learning Vim] was top priority on my list,” Jay reveals. “It was amazing.… I paired with people, and within two days I was exclusively in Vim.” He attributes his dramatic conversion to simple observation. “I could quickly absorb the little tips and tricks just by watching. Anytime I was pairing with someone using Vim I’d ask, ‘Hey, how did you do that?’” After a while, Jay found that he was teaching other programmers some of his own tricks. He proclaims, “The best way to learn stuff is to do it with other people.”

Jay’s giant steps toward mastering Vim are a great example of why it’s important to check your ego at the door and never be afraid to ask questions when pairing. “I went in with a very practical understanding of my ability,” he says. “I expected to be overwhelmed…but I was very surprised at how much I already knew.”

Ad hoc pairing is working for Jay’s team, but he still tries to introduce more structure when he can. “I like the idea of officially establishing the driver and navigator in each session,” he explains, “but we don’t usually do that.” The idea of having specific roles originated with colocated pair programming, and corresponds to various patterns often used to delegate responsibilities and keep a session’s momentum going. Although some of these are popular, they are equally despised by some teams.

Patterns of Pairing

In the early days of extreme programming, a number of patterns emerged that served to coordinate how two developers worked together. They probably came about because the concept of pairing was new, and most programmers weren’t used to anything other than working alone. Some of these patterns

deal with issues specific to colocated pairs, but a few have carried over into remote pairing.

Tag Team In this pattern, programmers take turns as the *driver*, who is in control of the keyboard, and the *navigator*, who contributes to the task verbally. The pair can alternate between these roles at preset time intervals or in an ad hoc fashion. In both cases, the driver writes code while the navigator acts as a reviewer and/or foreman. As reviewer, the navigator is responsible for identifying any mistakes the driver makes at the code level. As foreman, the navigator is responsible for thinking strategically about the overall structure of the code base and whether the code is solving the business problem at hand.[10]

Ping-Pong This pattern is an extension of both the tag-team pattern and test-driven development. The process begins with a programmer writing a test to define some new behavior or replicate a bug. The test will fail because it's new, at which point the first programmer passes control to the second programmer, whose job is to make the test pass. Throughout the process, both programmers communicate and discuss the problem, but control of the mouse and keyboard is strictly divided between the two phases.

Let the Junior Drive This pattern assumes one programmer is a novice and the other programmer is significantly more experienced. For the entire session, the more experienced programmer acts as the navigator while the less experienced programmer acts as the driver. It's important to note that some experts consider this an antipattern, as it can slow down the session by restricting fluid interaction between programmers. That might explain why studies conducted on real pairing sessions at large-scale companies find the boundaries between these roles break down in practice —resulting to an approach more similar to tag team than anything else.[11]

Parallel Pairing (aka Buddy Programming) In this pattern, two programmers work independently on the same problem and converge at the end to compare solutions and ultimately merge them. In this way, it mimics the difference between concurrency and parallelism in computer systems. Parallel pairing is not truly pairing, but it does have the advantage of eliminating many of the technical challenges described in this book because developers work independently for the most part.

10. *Pair Programming Illuminated [WK02]*
11. *Pair programming and the mysterious role of the navigator [BRd08]*

Trio Programming This includes any scenario in which more than two programmers attempt to work together on the same piece of code. Most of the tools discussed in this book, including tmux and VNC, support sharing with multiple users. In both cases, the host starts a session as normal while multiple clients connect to the session from distinct machines. As with the tag-team pattern, only one person acts as the driver during the session, but there will be multiple navigators. Thus, it's important to clearly define each person's role when implementing this pattern. Trio programming is often used when pair programming is strictly enforced (that is, solo programming is forbidden) but the team has an odd number of members. The disadvantage of this pattern is that each programmer gets less time to drive than would be the case with fewer people in the session. For this reason, many organizations discourage it. Joe Moore calls it an antipattern.

By most accounts, experienced pair programmers don't use these patterns—at least intentionally. The programmers at Test Double, Pivotal Labs, Big Nerd Ranch, and several other companies describe a typical pair-programming session as being largely unstructured. If they follow any pattern, it might best be described as an ad hoc tag-team pattern. These programmers are experienced enough at pairing that they don't need the rigidity of strict patterns. But there are a some exceptions.

"Ping-pong is great in interviews [and] in coaching situations," says Justin Searls. "In an environment where [no one] has done pairing before…[I'm] a little more formal and I start with ping-pong so that people have good prompts for what's next."

As you remote-pair-program, begin to use the ping-pong pattern. It will give you and your partner clear cues that help you learn when to switch roles. As you become more comfortable, move on to the tag-team pattern with preset intervals. Switch to the ad hoc tag-team pattern only when you begin to feel that you can comfortably anticipate the need to switch roles. You may end up loving or hating these patterns, but in either case you'll learn more about what works and what doesn't when pairing remotely. After time, you'll develop your own methods and patterns that work for you and your organization.

Wrapping Up

The majority of this book focuses on solving technical problems, but some of the most difficult remote-pairing issues are people problems. If you follow the advice of the programmers in this chapter you'll be well on your way to solving or even avoiding them entirely. But sometimes technology can solve people

problems, too. Your software stack can improve cooperation and make the remote-pairing experience smooth and comfortable. If you're happy with your tools, you'll probably be more productive.

Whatever tools you choose, remember that a human being sits at the other end of the connection. The same rules of etiquette that apply to colocated pair programming apply to remote pair programming. In fact, communication, sharing, and listening to your partner might be *more* important when pairing remotely.

Go pair with someone right now. Pick a project you know nothing about—try the Linux kernel.[12] Then grab a friend or post a tweet on Twitter with the #pairwithme hash tag. You can say something as simple as, "I want to learn about the Linux kernel. Anyone want to #pairwithme?" It's almost guaranteed that in the following thirty minutes you'll learn something new that improves your daily programming routine.

12. https://github.com/torvalds/linux