The Pragmatic Bookshelf

PragPub

The First Iteration

IN THIS ISSUE

* Pair Programming in a Flash
* Practical Mock Advice
* Language Lessons
* When Did That Happen?

Issue #24
June 2011

Contents

FEATURES

Pair Programming in a Flash 12
by Jeff Langr, Tim Ottinger

The rules of pairing, pairing smells, and what you can do when a pair isn't available.

Practical Mock Advice 17
by Zach Dennis

Zach shows mocks some much-needed respect.

Language Lessons 24
by Michael Swaine

A guide to all the programming language articles published in PragPub to date.

When Did That Happen? 30
by Dan Wohlbruck

The author of the first book on computer programming was born in this month, nearly a century ago.

DEPARTMENTS

Up Front 1
by Michael Swaine

Something old, something new...

Choice Bits 2
On a bus going to a secret location! Fueled by desperation! And a disturbing lack of pants.

Guru Meditation 4
by Andy Hunt

Learning curves are typically not uniformly steep, or we'd call them learning slopes. Andy tackles one of the steeper segments of the agile learning curve.

Way of the Agile Warrior 8
by Jonathan Rasmusson

Velocity is a powerful tool for planning. The trick is to get managers to use it correctly.

Calendar 33
Author sightings, upcoming conferences, and all the coverage of the royal wedding that you will ever need.

Shady Illuminations 38
by John Shade

John can't resist the temptation to make fun of dumb laws, short attention spans among technology marketers, and Mark Zuckerberg.

Published monthly in PDF, mobi, and epub formats by The Pragmatic Programmers, LLC, Dallas, TX, and Raleigh, NC. E-Mail support@pragprog.com, phone +1-800-699-7764. The editor is Michael Swaine (michael@pragprog.com). Visit us at http://pragprog.com for the lowdown on our books, screencasts, training, forums, and more.

ISSN: 1948-3562

Up Front

Permeable Boundaries

by Michael Swaine

Something old, something new...

Electronic publishing is permeable publishing.

Old-fashioned print-on-paper publishing wrapped each book or magazine in an impermeable membrane, the cover. Inside the cover, you were in the world of *Amazing Stories* or *Wuthering Heights*. Each book or magazine was its own world.

Electronic publishing, potentially at least, embeds a book or magazine in the real world. Or the World Wide Web, which is rapidly coming to mean the same thing, it seems.

An electronic magazine, like this one, can link to articles outside its covers. It can even link to articles in other issues of itself. So, as we do in this issue, we can include an additional two dozen articles that logically fit the theme of an article in this issue. And in doing so, we can present the material in a new context, as we do in "Language Lessons" in this issue.

But I don't want to give the impression that this issue is all *PragPub* Remixed.

Zach Dennis is here with some fresh insights on using Mocks. Zach helped write The RSpec Book [U1], so he knows his Mocks.

Jeff Langr and Tim Ottinger have been demonstrating what you might call pair authoring in *PragPub* recently, and they are back this month with the first part of a two-part exploration of pair programming.

And that's not all. Dan Wohlbruck flashes back to the first book on programming ever written, Andy Hunt tackles one of the steeper segments of the agile learning curve, and Jonathan Rasmusson devotes his "Way of the Agile Warrior" column to getting you up to speed on velicity.

As for John Shade, well, he has a few things he wants to get off his chest.

Welcome to our permeable little world. Oh, and one more thing: Next issue will be a bit special. We're going to focus on Clojure, a language that has permeable boundaries itself, with ties to one of the oldest programming languages and some of the most cutting-edge applications.

External resources referenced in this article:

[U1] http://pragprog.com/refer/pragpub24/titles/achbd/the-rspec-book

Pair Programming in a Flash

How to Pair, and Why

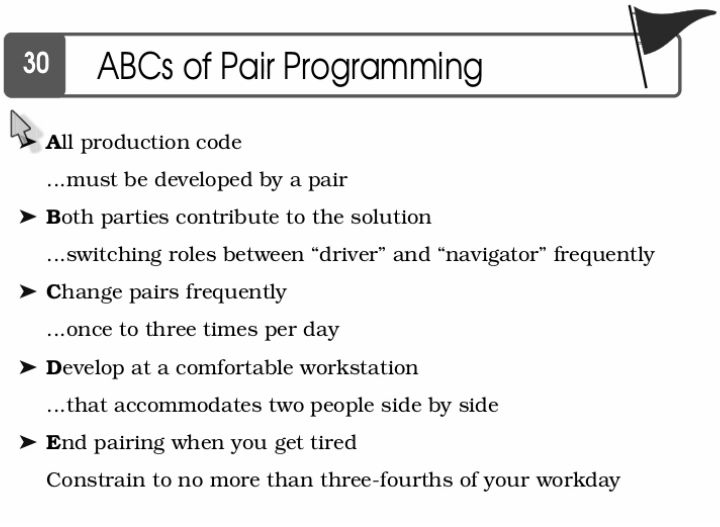
by Jeff Langr, Tim Ottinger

Jeff and Tim enjoy pairing so much that they want you to learn how to do it well. Here they present three cards on pairing from their Agile in a Flash [U1] card deck, covering its fundamental rules, a number of smells for you to sniff out, and guidelines for what you can do when a pair isn't available.

Yes, we're going to resurrect this contentious practice that divides programmers into oil-and-water camps. As much as some developers can't stand the idea of pairing, we continue to find value and enjoyment in it, hence our insistence on revisiting it.

The Rules of Pairing

Most developers are at least familiar with the concept of *pair programming*, or pairing, but let's quickly review the ground rules. They seem brief and simple:



We didn't dream up these rules for Agile in a Flash [U2] just to fill out the letters from A through E! We learned these more-or-less original guidelines for pairing as part of extreme programming. More than a decade later, after observing many development teams adopt and apply them (or not!), we see little need for these core rules to change. We've also noticed that teams that attempt pairing and then subsequently give up have usually violated one or more of these rules.

Yes, rules are meant to be broken—particularly as you advance to the *shu-ha-ri* mastery phase of *ri*—but *shu*'ers and *ha*'ers should first obtain a solid understanding of pairing by following all of the rules. Here's why.

- *All production code must be done by a pair*. You build production code with a pair to avoid institutionalizing low quality code in your system. Once bad code gets in, it increases the cost of maintenance on your system. It's also expensive to remove. You could use after-the-fact review to sidestep this #1 rule for pairing, but these reviews incur a high cost for what are typically only small improvements to the quality of the overall solution.

- *Both parties contribute to a solution*. Pairing is a social activity, which is the primary reason it's a challenging practice. Think of a pairing session as a collaborative design session during which you capture the result in code. You're continually discussing the design—sometimes vocally, sometimes in code. You're not taking turns watching someone else code—that would be pointless!
- *Change pairs frequently*. Creating quality software requires socializing the code and solutions throughout the team. Stagnant pairings can result in solutions that are almost as bad as those produced by lone, unchecked developers. Try to take the time to swap in a new set of eyes for each solution. Not only will bringing on a newcomer bring the quality of any given solution up a notch, an effective context switch will also require an improvement in the code's readability.
- *Develop at a comfortable workstation*. A successful social activity must be comfortable for all involved. Discomfort will dissuade people from wanting to pair. Neither partner should be disadvantaged by the space, and adding large monitors and good chairs make a bigger difference than you'd expect.
- *End pairing when you get tired*. Tired developers make more mistakes and derive less effective solutions, pairing or not. But a good pairing session sucks you in while time flies by, sometimes making it hard to notice that your brain is operating at less than optimal capacity.

Pairing Smells

- Unequal access
- Keyboard domination
- Unhealthy relationships
- Worker/rester
- "Everyone does their own work"
- Endless debate

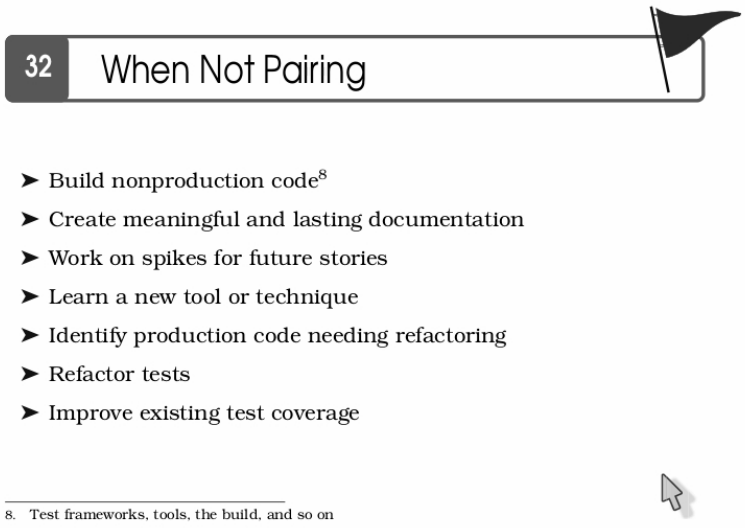
Pairing Smell:	Unequal access
Relates to:	D, Develop at a comfortable workstation
What to Do:	Make sure that both developers can easily get to the keyboard and see the monitor clearly, without having to play musical chairs or perform acrobatics!
Pairing Smell:	Keyboard domination
Relates to:	B, Both parties contribute to a solution

What to Do:	Ensure that keyboard control passes easily between both programmers. Talk about what you're doing, and learn to communicate in code as well as spoken word.
Pairing Smell:	Pair marriage
Relates to:	C, Change pairs frequently
What to Do:	It's easy to fall into the habit of working with one or two people that you know better than the rest. Long-term, though, it's far more valuable to be able to understand and collaborate with anyone on your team.
Pairing Smell:	Worker/Rester
Relates to:	A, All production code is produced by a pair
and:	B, Both parties contribute to a solution
What to Do:	We all get tired or frustrated with a task from time to time, but a better solution than sitting back and disengaging is to be honest and request a break or switch pairs.
Pairing Smell:	Second Computer
Relates to:	B, Both parties contribute to a solution
What to Do:	While some expert practitioners have found using two computers effective, it's too easy for a second computer to be a distraction for the person not currently coding.
Pairing Smell:	"Everyone does their own work"
Relates to:	A, All production code is produced by a pair
and:	B, Both parties contribute to a solution
What to Do:	When a manager mandates individual accountability, the interest in "getting my stuff done" can result in me giving short shrift to "your stuff," and thus reducing the overall quality of the system.
Pairing Smell:	"90% of work 90% done."
Relates to:	B, Both parties contribute to a solution
What to Do:	Stories that aren't completely implemented are a result of individual interest in getting their tasks done, and not in producing a combined solution.
Pairing Smell:	People who can't stand to program together
Relates to:	C, Change pairs frequently
What to Do:	Pairing or not, dissension on a team can devastate its ability to deliver. At least with pairing, the issue becomes obvious to any good coach or manager that intervention is required.

Pairing Smell:	Debates lasting more than 10 minutes without producing new code.
Relates to:	Well, this isn't a direct violation of any of the ABC's of pairing, but it does go against the core incremental nature of agile.
What to Do:	The more you learn to debate and demonstrate in code, the more you will be comfortable with taking demonstrably safe incremental steps (sometimes backward) to grow your system.

Sometimes the pair won't notice any of these problems; a coach or any external observer may have an easier time of spotting and suggesting solutions.

When Not Pairing



"But Tim and Jeff, we can't pair all the time. We have odd numbers. We have meetings where some people have to disappear for a while. Not everyone is in the office at the same time. And we really like to have good excuses for why we can't pair."

Each team and each day has its own challenges, but if you find yourself reaching for the excuses, perhaps your team isn't really interested in pairing. If you are facing *legitimate* difficulties that prevent you from pairing, you'll want to derive some ground rules for how to proceed. Producing production code alone should be a last resort, and if you check in to that resort, you need a backup contingency to deal with that new, potentially frightful code.

The "When Not Pairing" Agile in a Flash [U3] card recommends a number of useful tasks that can accelerate the whole team. Occasional breaks from pairing allow your team to put effort into improving things in your environment that might otherwise degrade. Build systems are an obvious target in most teams—how many times have you cursed about the complexity of the build, or the fact that it spews fifty pages of mostly worthless material that buries questionable exceptions?

Not everyone wants to pair 100% of the time, either. Jeff looks forward to having some time each week—even if it's only a couple hours—to do things

like play around with alternate solutions for an upcoming story, to experiment with a new API, or to clean up tests written earlier.

Programming solo? Just remember that there are always plenty of things you can do that will benefit the rest of the team.

But Why?

We pair because we enjoy it. Recently, Jeff moved into a development role where he found himself programming alone in an office for the bulk of the day. That's nothing new—he spent the first 18 years of his career not pairing and often isolated in a cube or office. Now that he's returned to solo development, though, he yearns for the social interaction of a pair and the ability to immediately bounce ideas and questions off someone else. It's not that he's incapable of developing good software on his own, but he feels much more effective when working as half of a programming pair.

Is our enjoyment enough to justify the cost of doubling up developers? Tim's recently been concerned that we spend too much time talking about *personal enjoyment* in the office space—no doubt a reaction to some folks' insistence that "this is all serious stuff and money we're talking about. The business doesn't care whether or not you're happy, you should just be happy to have a job." Harumph. Burnout, employee turnover, and the low-quality product of disgruntled workers is something the business should be dearly concerned with.

In any case, there are many reasons to embrace pairing: Benefits can accrue for programmers, their managers, and for the business as a whole. We'll talk about those benefits of pairing next month.

About Jeff

Jeff Langr has been happily building software for three decades. In addition to co-authoring Agile in a Flash [U4] with Tim, he's written another couple books, *Agile Java* and *Essential Java Style*, contributed to Uncle Bob's *Clean Code*, and written over 90 articles on software development. Jeff runs the consulting and training company Langr Software Solutions from Colorado Springs.

About Tim

Tim Ottinger is the other author of Agile in a Flash [U5], another contributor to *Clean Code*, a 30-year (plus) software developer, agile coach, trainer, consultant, incessant blogger, and incorrigible punster. He writes code. He likes it.

Send the authors your feedback [U6] or discuss the article in the magazine forum [U7].

External resources referenced in this article:

[U1] http://www.pragprog.com/refer/pragpub24/titles/olag/Agile-in-a-flash

[U2] http://www.pragprog.com/refer/pragpub24/titles/olag/Agile-in-a-flash

[U3] http://www.pragprog.com/refer/pragpub24/titles/olag/Agile-in-a-flash

[U4] http://www.pragprog.com/refer/pragpub24 /titles/olag/Agile-in-a-flash

[U5] http://www.pragprog.com/refer/pragpub24/titles/olag/Agile-in-a-flash

[U6] mailto:michael@pragprog.com?subject=agile-reflections

[U7] http://forums.pragprog.com/forums/134