Pair Programming - External

At Focused Labs we default to practicing pair programming, so what this means is that the majority of your engineering time is spent pairing. However, there are circumstances such as an odd team number that would allow for some soloing. Below, you will find resources and information about why and how we do it.

> Core Collaboration Practice

Pairing as a practice has existed in the industry for a while and means many things to many people. This article is super in-depth, https://martinfowler.com/articles/on-pair-programming.html.

This notion page covers what pairing means to Focused Labs and why it is one of our core practices.

What is pairing to us?

It's our default

It's not just for blockers anymore!

Pairing here consists of two equal partners writing code together: one computer, but two of everything else (monitors, keyboards & people). After stand, each team determines their pairs for the day. We try to rotate our pairs daily unless something that requires a lot of context is in-flight. Throughout the day, we strive to help each other grow our skills and share knowledge about the codebase and tools we're using.

While pairing is our default, it's a flexible practice. Sometimes, you wind up with an odd-numbered team, so you solo for a day. Sometimes, you might be on a one-pair team, and rotation isn't an issue. We might adjust the practice to current conditions ⛈️, but it's a baseline practice here and something you should expect to spend a fair amount of time doing.

Pairing isn't only, a practice, it's also a skill. Being a good pair involves being able to participate in constant feedback and collaboration, for more tips on being a good pair, keep on reading!

What is it NOT?

Code Reviews

Code review is a hard practice for little return. We have found pairing to be *far* more effective, especially for long term sustainability.

Occasionally for hard problems

- Pairing is a skill and like any skill, you can only get better at it by doing it.
- If you are working long term on a significant amount of code that is rote, your codebase is probably missing an abstraction. A pair will help you find the appropriate abstractions to avoid this in the future.

Why default to pairing?

Pairing can improve quality of life as a developer. Sometimes writing code solo can be fun and feel productive, but sometimes... it can go sideways. We've been there: rabbit-

holed for hours on something that was essentially a typo. The recipient of a PR review filled with comments on using `const` vs `let` ...while leaving the architecture change you'd like feedback on unmentioned. Receiving feedback on a feature that you'd spent hours on, and feeling personally attacked or like an impostor. Pairing is great for preventing too-deep-in-the-weeds problems, because you've got someone there to pull you out, and a second set of eyes to see what you can't. You can get meaningful feedback on what you're writing from someone who has shared context and shared concerns. Finally, because responsibility for the code is distributed across the team, we can still have pride in crafting something well, without that feeling of ownership becoming possessive or defensive. Pairing distributes the burden and shares the joys.

Pairing is great for building teams. By working side-by-side with someone, you're encouraged to reconcile their way of thinking with your own. Getting out of your own head like this can strengthen everyone's understanding of the task at hand, and fosters in-depth communication. By switching pairs frequently, no one person holds all the context, and knowledge is distributed across the team. In the event that you're onboarding to a new project, or have less experience in a given language or platform, pairing offers valuable learning opportunities beyond what the docs can provide. And as the more experienced pair, it offers a chance to learn how to communicate your knowledge plainly (which is a valuable skill for consultants); also, nothing cements what you think you know, like explaining it to someone else.

The act of talking through any given feature optimizes the code in a few ways. Communicating often requires clarifying your own thought process and getting immediate feedback on assumptions and understanding. In making sure that the code that we're pairing on is clear to each other, the code becomes more readable for others and our future selves. Given a diverse team, people come from different backgrounds (whether education, workplace, language preference or just personality), and that diversity gets passed along to the code's style and approach. Plus, two sets of eyes means fewer bugs.

Pairing is just good practice. There are a few:

- Fosters empathy
- You involve people in the design rather than the output
- Fast feedback loops

- Less possessiveness makes introducing client devs to the codebase easier

How to be a good pair?

We only get the benefits of pairing when we're being the best pair we can be.

All around good advice:

- Listen first
- Take breaks: frequent breaks are key to regaining focus, feeling better and being able to think on something in your own way

If you're the confident type...

- Let your pair to have equal time on the keyboard, and encourage them to drive if they're hesitant to take the keyboard
- If you're the person with more context/experience, bring your pair up to speed and don't just forge ahead without them

If you're the introverted or struggles-with-impostor-syndrome type...

- Speak up: If you're feeling lost or out of context, don't be afraid to raise it. Calling out confusing code helps to make sure that it's written more clearly.
- *[ADD MORE]*

Tools!

- Tuple
- Screen