How to Pair Program

Author Info
Last Updated: June 4, 2020

Pair programming is a method of programming in which two people work together at one keyboard. One person, "the driver", types at the keyboard. The other person, "the observer" (or "navigator") reviews each line of code as it is typed, checking for errors and thinking about the overall design.

Some benefits you can expect: better code (simpler design, fewer bugs, more maintainable), higher morale (more fun!), shared knowledge throughout your team (both specific knowledge of your codebase and general programming knowledge), better time management, higher productivity.

Steps

1 Start with a reasonably well-defined task before you sit down. The task should be something you are confident that you can complete in an hour or two. For example, "Add 'maintenance history' to the moving-van database code." You may find it helpful to outline what you plan to do before you begin to code.

2 Agree on one tiny goal at a time: something you can complete within a few minutes. Stating the problem in words to another person helps focus your mind and helps engage your partner's mind. It also ensures that you both know what you are working on right now.

3 Rely on your partner, support your partner.

- When you're the driver, complete the current tiny goal as quickly as you can, ignoring larger issues. Trust the observer to be your safety net.
- When you're the observer, read the code that the driver is writing as he or she writes it. Your job is code review. You should pay total attention, aiming to let nothing get by you. Think about possible bugs, larger issues, and ways to simplify or improve the design. Bring up errors and code that you find unreadable right away. Wait until the current tiny goal is done to bring up larger issues and ideas for design improvement. Jot these later tasks down so the driver can stay focused on the present tiny task. For example, if you see that the current code fails to account for a null input, write down on a piece of paper, "Add unit test for null input."
- When you're the observer, don't dictate the code. The driver should be actively thinking about how to achieve the current task, not just typing passively. And as the observer, you should exploit the fact that you don't need to invent the small details; you can and should think at a higher level. Saying "That looks right. How about handling the case where we're passed a null pointer now?" is better than "OK, now type 'if (s == NULL) { return ...'"

4 Talk a lot! Say what you are about to do, ask for an implementation idea, ask for a better way to solve the problem at hand, bring up alternative ideas, point out possible inputs that the code doesn't cover, suggest clearer names for variables and subroutines, suggest ways to implement the code in smaller steps, tell the driver that little bit of API knowledge that they need right at the moment they need it, etc. Listen a lot, too, of course. When people are pairing well, they are talking back and forth almost non-stop. Here are some common things to say while pairing:

- "Do you think this is a valid test?"
- "Does that look correct to you?"
- "What's next?"
- "Trust me" (when it's easier to write a little code to make your point than to say it out loud)

5 Sync up frequently. As you are working together, you will find yourself getting out of sync: becoming unsure what your partner is doing, or becoming unclear about the current task. This is normal. When it happens, sync up again. The key to good pairing is to sync up very frequently—within seconds or a minute of noticing that you're out of sync. If you are spending five minutes (or more) out of sync, you might as well be coding solo, because it's the frequent re-syncing that creates the

- When you can, say what you are about to do before you do it. Better yet, ask your partner; for example, "Shall we write the test for the null case now?" Sometimes, though, you have to write code in order to understand your thought, and that's okay. Then you can say you are doing that: "I need to type this to see if it's a good idea." Best to keep that kind of exploration to less than a minute, though.
- When your partner asks if you agree with something, like "Shall we write the test for the null case now?" or "I think this method can be deleted now. Do you agree?", say "Yes" or "No" clearly and immediately.
- It's okay to pass the keyboard back and forth very frequently. For example, sometimes it's much easier to "say" something by typing it in code than by trying to explain it out loud. So let the observer grab the keyboard and type. Then you can switch back, or let the observer keep driving, whichever makes more sense right then.

6 Take a moment to celebrate as you complete tasks and overcome problems. For example, each time you get a test to pass, give each other a high five. If you also high-five each time you get a new test to fail, you'll really get into the groove of collaborative programming and test-driven design.

7 Switch roles often—at least every half hour. This keeps you both fully engaged, both of you in tune with the low-level details and the big picture. Also, driving full-blast can tire you out, and it's hard to maintain the vigilance required of the observer role for longer than half an hour. Switching roles recharges you.

Tips

- Be especially courteous. For example, when your partner points out an error, say, "Thank you." When pointing out an error, do so gently, to avoid offending egos. Errors and correcting them are a normal part of programming, not evidence that someone lacks ability. As the observer, let the driver finish writing a complete line of code before pointing out an error. Most people find it annoying to be corrected as they type, but helpful to be corrected when they're finished typing.

- Laptop computers are less than ideal due to the small size and difficulty seeing the screen from an angle, but you can still pair at a laptop. One trick is to have the observer sit somewhat behind the driver rather than to the side.

- Try using two keyboards and two mice. "Switching" between drivers becomes much easier. This is accomplished with a device called a KVM Switch, which you plug the keyboards and mice into, and run the wires to the computer. The switch is as easy as turning a knob. A second monitor can be added just as easily, and some may not support a monitor (although the "V" stands for video) and may be ideal if your needs are less.

- The difference between okay/poor pairing and wonderful pairing is simply this: Pay attention. Look at the screen as the driver types. When your partner speaks, listen. Don't eat cereal. Don't check your phone. Don't check reddit. Let no detail get past you. When you pay attention, talk, and listen, you effortlessly become engrossed in the work, and you enter the joyful "flow" of pairing. This is when your productivity skyrockets. If one or both of you is mentally checking out, then you might as well be working solo.

  - If you have a lifelong habit of mentally checking out when other people are talking to you, or always multitasking between 5 different things, try putting 100% of your attention on the task at hand as a one-day experiment. See for yourself if you enjoy it and like the results you get. You might mentally check out repeatedly, just out of habit. If that happens, your pair can gently remind you of the current task each time. Getting focused can be much easier with a pair than alone.

- Don't argue about either architectural matters or trivial matters like how far to indent the braces. These things should be decided on before you pair.

- Write a unit test first, before you write the implementation (if the code is unit-testable). The unit test helps define the next tiny goal in a way that you both understand, since you can both see the code. The next tiny goal becomes, "Make this test pass." Test-driven development can be tough to learn at first, but pairing with someone who's already accustomed to it can teach the skill pretty quickly.

- Sit at a table where you can easily pass the keyboard between you to switch roles. A plain ol' rectangular table works best (as opposed to weird, curvy, "ergonomic" furniture).

- The person who knows less about the system or language should do most of the driving, to ensure that the novice stays engaged. You learn more actively than passively.