We're Better Together: Pair Programming

So far, you've been solving problems and writing code on your own. You'll certainly spend a lot of time doing this in your future, but why not try a better way? Let's discuss *pair programming*: an alternative approach that can boost your output, reduce errors, and improve your mood all at the same time!

We'll cover:

- a brief history of team/pair programming,
- typical roles when pairing up,
- and benefits of learning this useful skill!

Team mentality

Despite what "hacker" personalities in movies might have you think, software development has never been a single-player game. Tracing back to the earliest days of computing, computer scientists have worked in pairs or teams to solve the biggest problems. Fred Brooks, in his popular software/project management guide "The Mythical Man Month" (published in 1975) even advocated for small"surgical teams"to approach software development!

Even so, it's easy to get focused on a problem and stop communicating with your teammates. To prevent this behavior, many teams practice *pair programming*, an approach in which two developers work together on a single computer. Pairing up ensures that both developers are engaged with each other and with the problem at hand, and it helps us share knowledge faster!

Pair programming is by far the most popular approach, but there are other ways for small groups to collaborate! You may also hear about *mob programming*, where a group of 3+ individuals gather around a single screen to work through an issue, or *extreme programming (XP)*, a highly-structured approach to pair programming where the whole team rotates through projects. We'll refer to all of these multi-developer approaches as *collaborative programming*.

There are a few common concepts in all these approaches:

- **One shared device for coding:**Keeping your code in sync with someone else is a bigger challenge than it's worth, so collaborative programmers have found that coding only on one device keeps it simple. You can always share your code with each other once you're done working on it.

- **Everyone has a job:**Regardless of how many people you're collaborating with, it's important that each person has a role to play. This increases engagement and ensures that no one gets bored! If you find yourself in a group but without a clearly-defined role, there may be a better use of your time.

- **Everyone gets a turn:**The best way to learn is by doing, so everyone should be able to work in all the roles on a collaborative programming team. This means developers may rotate into different roles on a timer, or change roles with each new session.

- **No one is "too good to pair":**It's important to note that pairing should be a **whole team**process. One of the benefits of pair programming is that a more senior engineer can share knowledge with a junior engineer, skilling them up faster. No truly collaborative programming session excludes certain people because they're outliers. It's never too early (or too late!) to start pairing up.

Pair Programming Roles

When we talk about pair programming specifically, there are two well-known roles: that of the *Driver*and that of the *Navigator*. Let's break down their individual responsibilities.

The Driver

The *Driver*takes ownership of the keyboard. They'll be in charge of typing code and asking questions. The driver should focus on the current task and can let go of the bigger picture for a little while. The Driver might also suggest ways to improve/refactor the code, as the project continues.

The Navigator

The *Navigator*is in charge of what's being typed and maintaining the project's momentum. They should lead the discussion and direct the driver about what to type. The navigator won't do any typing themselves. Instead, their goal is to make it as easy as possible for the driver to create code. While the driver types, the navigator should be double-checking the code for errors.

Both roles

If these roles sound limiting to you, good; that's the point! Having a clearly defined role means you never have to wonder what you should be doing at any given time. If you're driving, you should be writing code. If you're navigating, you should be designing what comes next and sharing that with the driver.

These roles provide some structure, but they're not excuses for poor communication. It's up to **both**programmers to discuss the plan early and often. As the navigator leads the way, the driver should be constantly second-guessing the plan. If there's a potential problem, it's okay for the driver to bring this up! This is one of the strengths of pair programming: both developers have a voice in service of the final product.

Both the driver and navigator should help keep each other on task as well. If you find yourself getting quiet or distracted, switch roles! The change of context should help you get your head back in the game.

Why pair up?

This all sounds like a lot of work! If you're already capable of coding alone, why might you want to pair up at all? Let's look at some of the benefits.

First, some statistics. A study from 2000found that pair programming results in 15% slower development time. Yikes! That sounds a lot less scary when you recognize that we would theoretically expect it to result in 50% slower time, though. After all, we're taking two people who could be working at "full speed" and assigning them to one task. Somehow, this process only causes them to move 15% slower - wow!

The same study notes the payback for that loss of velocity: the resulting code contains 15% fewer errors. If this sounds like an even trade, consider this: debugging, correcting, and redeploying that erroneous code could take from 15 to 60 times as long as the paltry 15% slowdown caused by pairing up! Spending a little extra time up front can make a world of difference later on.

Pair programming is also a great way to fulfill your social needs. Coding alone can be a lonely endeavor, especially when you face new challenges and aren't sure how to proceed. Instead of getting lost in a search engine, pairing partners can help each other navigate these challenges and celebrate each others' successes.

Lastly, a big motivator for being a practiced pair programmer is the job search. Many companies include pair programming in their interview process. You may be asked to code while some of the company's engineers observe, or you might get to work directly with another engineer on a collaborative project! Practicing this skill now means you'll be comfortable when the stakes are higher, and more likely to make a good impression on the interviewing team.

What we've learned

When it comes to writing code, we are truly better together! Pair programming is an effective process for writing more code, writing better code, and improving your development skills at a dramatically increased rate. Starting now, we'll be encouraging you to pair with other students on App Academy projects.

After reading this lesson, you should feel comfortable:

- defining collaborative programming,
- describing the roles of the Driver and Navigator,
- and naming a few benefits of pair programming.