You Are Not Your Code: Empathetic Communication In Engineering

We've presented a rigid process for pair programming, but we left out a big part: **communication**! While we've mentioned that communication is important, we haven't really dug into the value of this skill. Let's discuss some communication approaches that help you maximize your time when pair programming

- and collaborating in general!

We'll discuss:

- empathetic communication when working on code,
- how to appropriately critique the work of others,
- and how to receive & respond to criticism of your own work.

Understanding code-centric vs. human-centric language

The code you write can feel deeply personal. You've put time into it, refactor after refactor, honing it to be both highly efficient and easily understood. It's no wonder, then, that criticism of that code can feel personal as well! A common problem on software teams is the struggle between improving code quality without offending the programmers who wrote it.

When communicating about software, it can be helpful to use *code-centric*language. Here's an example:

```
// Human-centric
"You wrote an infinite loop.
If you do that again, you will crash the server."

// Code-centric
"The code submitted for issue #3 resulted in an infinite loop.
This behavior could crash the server.
What can we do to prevent that in the future?"
```

Notice how the code-centric example focuses on outcomes and looks towards improvement, instead of focusing on assigning blame and enumerating negative consequences. It can feel like a nitpick, but depersonalizing discussions of code quality in this way helps you focus on the bigger picture: your team and the product you're working on together.

It's important to remember that what we're **not**doing with code-centric communication is removing responsibility. As a developer, you may write code for use in critical industries like healthcare, finance, or defense. It's your responsibility to be constantly learning and growing to ensure that what you create is safe and correct.

However, unless you are a solo freelance developer working on a solo project, your team will share your goals and should be focused on uplifting each other. Having a positive outlook, even in the face of human error, can be the difference between a successful team and a team destined for failure.

A great practical example of code-centric language in action is this report by GitLabafter an accidental deletion of data. The company used this as a learning opportunity, improved their release tooling to prevent accidents in the future, and shared everything they learned with the community.

Applying empathy to pair programming

When working directly with a partner, the relationship is close so the stakes are high. It's important to maintain that relationship, but it's also important to

hold each other to a high standard! We can do both these things with *empathetic communication*.

Being *empathetic* means being considerate of the feelings of others. It's not just about recognizing challenges - it's important to relate those challenges to your own experiences and respond in a way that acknowledges them appropriately.

A common phrase you'll hear when discussing communication in software development is **"You are not your code"**. This is a simplification of the human- vs. code-centric communication styles we discussed. Keeping this separation between your partner (or yourself!) and the code you're producing together can help you communicate in a way that improves outcomes for everyone.

Let's look at another example. Imagine you're acting as navigator during a pairing session and reviewing the code your partner has typed.

```
// Human-centric
"Why did you you write the function the wrong way?"

// Code-centric
"Is this an alternative syntax for a function?"
```

We're not ignoring what might be a problem, but we are remaining open to a potential learning opportunity and drawing attention to it if needed. This difference in approach embodies the spirit of the "You are not your code" motto.

The context of "You are not your code" is helpful when receiving criticism as well. Remember during pair programming or code review that we're all on the same team: making better software! It can be tough to have your work critiqued, but it's the best way to improve. If you find a particular note affects you personally, try rewriting it for yourself using code-centric language.

When empathy goes awry

You may sometimes encounter other engineers who use "you are not your code" as a license to insult. The goal of this phrase is to encourage programmers to act with empathy. It's **never** an excuse to belittle someone with the caveat of "You can't be offended - I'm just talking about your code!". Notice that even that defense uses human-centric, personally challenging language.

Practice using "you are not your code" in a positive context whenever you can, and encourage others to do the same. Strive to treat the people around you in the same way you'd like to be treated. Remember that you're working with humans from many different backgrounds and with unique experiences, and that you should place respect for those human experiences over code quality, words-per-minute, or browser preference. That's true empathy.

Applying empathy to your code

Finally, remember that the code you're writing will likely be read and used by other developers in the future. Just like practicing empathetic communication in-person helps build relationships, practicing empathetic coding helps build good software.

Empathetic coding means thinking of those who will read or improve your code once you've moved on to other projects. Leaving clear comments and taking time to keep your code organized is one way to code with empathy. Another is working to use inclusive language in your test cases, or avoiding hard-to-understand single letter variable names.

Anything you can do to make the next developer's life easier is worth the time. Who knows - it might be **you** coming back to this project many months from now!

Why communication matters

So why does this all matter? App Academy is a software development program, not a public speaking course! Look a little deeper, though, and you'll see a wealth of benefits from improving your communication.

Selfishly, good communication helps you succeed. Lots of programmers with strong technical skills will apply, but those who can communicate clearly and efficiently will get the job. The relationships you build by being empathetic with your coworkers will also help you with positive references and opportunities for promotion!

Thinking more globally, it's important to remember that the tech industry is growing at a rapid pace. Accusatory communication is just one of many ways of*gate-keeping*, or making our industry harder to get involved with. This isn't always intentional, so improving your own communication skills is a great way to become more aware and ensure you're part of the solution, not part of the problem.

What we've learned

When programming, you're not just communicating with your computer! Practicing empathy in **all**your communication will result in a great return for you, your co-workers, and the tech community at large.

We hope you're now able to:

- differentiate between human-centric and code-centric language,
- articulate the meaning of "You are not your code",
- and list the benefits of practicing empathetic communication.