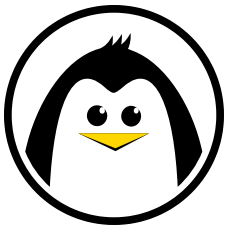

ENDLER.DEV

Open-source maintainer,
speaker, rebel. About me.

Published on 4th of April, 2025 · Updated on 7th of April, 2025

The Best Programmers I Know

Tagged with dev culture

I have met a lot of developers in my life. Lately, I asked myself: "What does it take to be one of the best? What do they all have in common?"

In the hope that this will be an inspiration to someone out there, I wrote down the traits I observed in the most exceptional people in our craft. I wish I had that list when I was starting out. Had I followed this path, it would have saved me a lot of time.

Read the Reference

If there was one thing that I should have done as a young programmer, it would have been to *read the reference* of the thing I was using. I.e. read the Apache Webserver Documentation, the Python Standard Library, or the TOML spec.

Don't go to Stack Overflow, don't ask the LLM, don't *guess*, just go straight to the **source**. Oftentimes, it's surprisingly accessible and well-written.

Know Your Tools Really Well

Great devs understand the technologies they use on a **fundamental level**.

It's one thing to be able to *use* a tool and a whole other thing to truly *grok* (understand) it. A mere user will fumble around, get confused easily, hold it wrong and not optimize the config.

An expert goes in (after reading the reference!) and sits down to write a config for the tool of which they understand every single line and can explain it to a colleague. That leaves no room for doubt!

To know a tool well, you have to know:

- its history: who created it? Why? To solve which problem?
- its present: who maintains it? Where do they work? On what?
- its limitations: when is the tool not a good fit? When does it break?
- its ecosystem: what libraries exist? Who uses it? What plugins?

For example, if you are a backend engineer and you make heavy use of Kafka, I expect you to know a lot about Kafka – not just things you read on Reddit. At least that's what I expect if you want to be one of the best engineers.

Read The Error Message

As in **Really Read the Error Message and Try to Understand What's Written**. Turns out, if you just sit and meditate about the error message, it starts to speak to you. The best engineers can infer a ton of information from very little context. Just by reading the error message, you can fix most of the problems on your own.

It also feels like a superpower if you help someone who doesn't have that skill. Like "reading from a cup" or so.

Break Down Problems

Everyone gets stuck at times. The best know how to get unstuck. They simplify problems until they become digestible. That's a hard skill to learn and requires a ton of experience. Alternatively, you just have awesome problem-solving skills, e.g., you're clever. If not, you can train it, but there is no way around breaking down hard problems. There are problems in this world that are too hard to solve at once for anyone involved.

If you work as a professional developer, that is the bulk of the work you get paid to do: breaking down problems. If you do it right, it will feel like cheating: you just solve simple problems until you're done.

Don't Be Afraid To Get Your Hands Dirty

The best devs I know read a lot of code and they are not afraid to touch it. They never say "that's not for me" or "I can't help you here." Instead, they just start and learn. Code is *just code*. They can just pick up any skill that is required with time and effort. Before you know it, they become the go-to person in the team for whatever they touched. Mostly because they were the only ones who were not afraid to touch it in the first place.

Always Help Others

A related point. Great engineers are in high demand and are always busy, but they always try to help. That's because they are naturally curious and their supportive mind is what made them great engineers in the first place. It's a sheer joy to have them on your team, because they are problem solvers.

Write

Most awesome engineers are well-spoken and happy to share knowledge.

The best have some outlet for their thoughts: blogs, talks, open source, or a combination of those.

I think there is a strong correlation between writing skills and programming. All the best engineers I know have good command over at least one human language – often more. Mastering the way you write is mastering the way you think and vice versa. A person's writing style says so much about the way they think. If it's confusing and lacks structure, their coding style will be too. If it's concise, educational, well-structured, and witty at times, their code will be too.

Excellent programmers find joy in playing with words.

Never Stop Learning

Some of the best devs I know are 60+ years old. They can run circles around me. Part of the reason is that they keep learning. If there is a new tool they haven't tried or a language they like, they will learn it. This way, they always stay on top of things without much effort.

That is not to be taken for granted: a lot of people stop learning really quickly after they graduate from University or start in their first job. They get stuck thinking that what they got taught in school is the "right" way to do things. Everything new is bad and not worth their time. So there are 25-year-olds who are "mentally retired" and 68-year-olds who are still fresh in their mind. I try to one day belong to the latter group.

Somewhat related, the best engineers don't follow trends, but they will always carefully evaluate the benefits of new technology. If they dismiss it, they can tell you exactly *why*, when the technology would be a good choice, and what the alternatives are.

Status Doesn't Matter

The best devs talk to principal engineers and junior devs alike. There is no hierarchy. They try to learn from everyone, young and old. The newcomers often aren't entrenched in office politics yet and still have a fresh mind. They don't know why things are *hard* and so they propose creative solutions. Maybe the obstacles from the past are no more, which makes these people a great source of inspiration.

Build a Reputation

You can be a solid engineer if you **do** good work, but you can only be one of the best if you're **known** for your good work; at least within a (larger) organization.

There are many ways to build a reputation for yourself:

- You built and shipped a critical service for a (larger) org.

- You wrote a famous tool
- You contribute to a popular open source tool
- You wrote a book that is often mentioned

Why do I think it is important to be known for your work? All of the above are ways to extend your radius of impact in the community. Famous developers impact way more people than non-famous developers. There's only so much code you can write. If you want to "scale" your impact, you have to become a thought leader.

Building a reputation is a long-term goal. It doesn't happen overnight, nor does it have to. And it won't happen by accident. You show up every day and do the work. Over time, the work will speak for itself. More people will trust you and your work and they will want to work with you. You will work on more prestigious projects and the circle will grow.

I once heard about this idea that your latest work should overshadow everything you did before. That's a good sign that you are on the right track.

Have Patience

You need patience with computers and humans. Especially with yourself. Not everything will work right away and people take time to learn. It's not that people around you are stupid; they just have incomplete information. Without patience, it will feel like the world is against you and everyone around you is just incompetent. That's a miserable place to be. You're too clever for your own good.

To be one of the best, you need an incredible amount of patience, focus, and dedication. You can't afford to get distracted easily if you want to solve hard problems. You have to return to the keyboard to

get over it. You have to put in the work to push a project over the finishing line. And if you can do so while not being an arrogant prick, that's even better. That's what separates the best from the rest.

Never Blame the Computer

Most developers blame the software, other people, their dog, or the weather for flaky, seemingly "random" bugs.

The best devs don't.

No matter how erratic or mischievous the behavior of a computer seems, there is *always* a logical explanation: you just haven't found it yet!

The best keep digging until they find the reason. They might not find the reason immediately, they might never find it, but they never blame external circumstances.

With this attitude, they are able to make incredible progress and learn things that others fail to. When you mistake bugs for incomprehensible magic, magic is what it will always be.

Don't Be Afraid to Say "I Don't Know"

In job interviews, I pushed candidates hard to at least say "I don't know" once. The reason was not that I wanted to look superior (although some people certainly had that impression). No, I wanted to reach the boundary of their knowledge. I wanted to stand with them on the edge of what they thought they knew. Often, I myself didn't know the answer. And to be honest, I didn't care about the answer.

What I cared about was when people bullshitted their way through the interview.

The best candidates said "Huh, I don't know, but that's an interesting question! If I had to guess, I would say…" and then they would proceed to deduce the answer. That's a sign that you have the potential to be a great engineer.

If you are afraid to say "I don't know", you come from a position of hubris or defensiveness. I don't like bullshitters on my team. Better to acknowledge that you can't know everything. Once you accept that, you allow yourself to learn. "The important thing is that you don't stop asking questions," said Albert Einstein.

Don't Guess

"In the Face of Ambiguity, Refuse the Temptation to Guess" That is one of my favorite rules in PEP 20 – The Zen of Python.

And it's so, so tempting to guess!

I've been there many times and I failed with my own ambition.

When you guess, two things can happen:

- In the **best case** you're wrong and your incorrect assumptions lead to a bug.
- In the **worst case** you are right… and you'll never stop and second guess yourself. You build up your mental model based on the wrong assumptions. This can haunt you for a long time.

Again, resist the urge to guess. Ask questions, read the reference, use a debugger, be thorough. Do what it takes to get the answer.

Keep It Simple

Clever engineers write clever code. Exceptional engineers write simple code.

That's because most of the time, simple is enough. And simple is more maintainable than complex. Sometimes it *does* matter to get things right, but knowing the difference is what separates the best from the rest.

You can achieve a whole lot by keeping it simple. Focus on the right things.

Final Thoughts

The above is not a checklist or a competition; and great engineering is not a race.

Just don't trick yourself into thinking that you can skip the hard work. There is no shortcut. Good luck with your journey.

Thanks for reading! I mostly write about Rust and my (open-source) projects. If you would like to receive future posts automatically, you can subscribe via RSS.

About me

I'm Matthias Endler, an open-source maintainer and Rust consultant at corrode, helping clients make their Rust code faster and more idiomatic.

I am the co-founder of Open Podcast, a pro-tool for podcast analytics (like Google Analytics, but for podcasts).

With a passion for linters and static code analysis, I also maintain analysis-tools.dev, an open platform with 600+ tools listed.

Prior ventures include Hello, Rust!, a YouTube channel about the Rust programming language and codeprints, a shop for getting art prints of your GitHub timeline.

For more info, check out my about page or the blog archive.

Support

Some links on this blog are *affiliate links* and I earn a small comission if you end up buying something on the partner site (at no extra cost for you), which I use to cover maintenance costs. I only link to books and resources that I'd personally recommend to friends.

Maintaining this blog and my projects is a lot of work and I'd love to spend a bigger part of my life writing and maintaining open source projects. If you like to support me in this goal, the best way would be to become a sponsor ❤️.