Chapter 1

Learn in Public

There are many principles offered in this Coding Career Handbook, but this is chief among them: **Learn in Public**.

1.1 Private vs Public

You have been trained your entire life to learn in *private*. You go to school. You do homework. You get grades. And you keep what you learned to yourself. Success is doing this better than everyone else around you, over and over again. It is a constant, lonely, zero-sum race to get the best grades. To get into the best colleges. To get the best jobs. If you've had a prior career, chances are that all your work was confidential. And of COURSE you don't share secrets with competitors!

Tech is a **fundamentally more open** industry. We blog about our outages. We get on stage to share our technical achievements. We even *give away* our code in open source. However, most developers act like tech is the same as every other industry. Most developers bottle up everything they learn in their heads, all the while hoping their careers grow linearly with years of experience at the *right* companies working on the *right* projects for the *right* bosses. Most developers strictly consume technical content without actually creating any themselves.

This is a perfectly fine way to build a career: ~99% of developers operate like this. Scott Hanselman calls them Dark Matter Developers — you can infer their presence from GitHub stars and package downloads, but it's hard to find direct evidence of their work. Their network mainly consists of current or former coworkers. When job hunting, **they start from zero every time**. They find opportunities only from careers pages or recruiters. To get an interview, they must serialize years of experience down into a one page resume by guessing employers' opaque deserialization and filtering algorithms. Then they must convey enough in 30-60 minute interviews to get the job. Even while on the job, picking up a new technology is a solitary struggle with docs and books and tutorials.

There is another way. You can Learn in Public instead.

What do I mean by learning in public? You share what you learn, as you learn it. You **Open Source your Knowledge** (Chapter 13). You build a public record of your interests and progress, and along the way, you attract a community of mentors, peers, and supporters. They will help you learn faster than you ever could on your own. Your network could be *vast*, consisting of experts in every field, unconstrained by your org chart.

When job hunting, prospective employers may have followed your work for years, or they can pull it up on demand. Or — more likely — they may seek you out themselves, for one of the 80% of jobs that are never published. Vice versa, you take much less cultural risk when you and your next coworkers have known each other's work for years. And when picking up a new technology, you can call on people who've used it in production, warts and all, or are even directly building it. They will talk to you — *because you Learn in Public*.

I intentionally haven't said a *single* word about "giving back to the dev community". Learning in Public is not altruism. It is not a luxury or a nice-to-have. It is simply the fastest way to learn, establish your network, and build your career. This means it is also sustainable, because you are primarily doing it for your own good. It just so happens that, as a result, the community benefits too. Win-win.

You never have to be 100% public. Nobody is. But try going from 0% to 5%. Or even 10%! See what it does for your career.

Tip: Some vulnerable people have personal safety or other reasons to *not* Learn in Public. These are totally valid. Since the majority of the time, we all are still **Learning in Private**, it's worth thinking about how to do that well too. Refer to Chapter 32 for a fuller discussion.

1.2 Getting Started

Make it a habit to create "**learning exhaust**" as a non-negotiable and automatic side effect of your own learning:

- Write demos, blogs, tutorials and cheatsheets
- Speak at meetups and conferences

- Ask and answer questions on Stack Overflow or Reddit
- Make YouTube videos or Twitch streams
- Start a newsletter
- Draw cartoons (people loooove cartoons!)

Whatever your thing is, **make the thing you wish you had found** when you were learning. Document what you did and the problems you solved. Organize what you know and then **Open Source Your Knowledge**.

You catch a lot of friends when you are **Helpful on the Internet**. It is surprisingly easy to beat Google at its own game of organizing the world's information. Even curating a structured list of information is helpful. I once put together a list of Every Web Performance Test Tool on a whim and it got circulated for months! People reshared my list and even helped fill it out.

> "But I'm not famous, nobody will read my work!" — *you, probably*

Don't judge your results by retweets or stars or upvotes — just talk to yourself from three months ago. Resist the immediate bias for attention. **Your process needs to survive *regardless* of attention**, if it is to survive at all. Eventually, they will come. But *by far* the biggest beneficiary of you helping past you, will be *future you*. If (when) others benefit, that's icing on the cake.

This is your time to suck. When you have no following and no personal brand, you also have no expectations weighing you down. You can experiment with different formats, different domains. You can take your time to get good. Build the habit. Build your platform. Get comfortable with your writing/content creation process. Ignore the peer pressure to become an "overnight success" — even "overnight successes" went through the same thing you are.

I get it: We all need feedback. If you want guaranteed feedback, **Pick Up What Others Put Down** (Chapter 19). Respond to and help your mentors on things they want, and they'll respond to you in turn. But sooner or later, you'll have to focus on your needs instead of others. Then you're back to square one: having to develop **Intrinsic Drive** instead of relying on External Motivation.

1.3 But I'm Scared

Try your best to be right, but don't worry when you're wrong. Keep shipping. Before it's perfect. If you feel uncomfortable, or like an impostor, good. That means you're pushing yourself. Don't assume you know everything. Try your best anyway and let the Internet correct you when you are inevitably wrong. Wear your noobyness on your sleeve. Nobody can blame you for not knowing everything. (*See **Lampshading**, Chapter 34, for more*)

People think you suck? Good. You agree. Ask them to explain, in detail, why you suck. Do you want to feel good or do you want to **be** good? If you keep your identity small and separate your pride from your work, you start turning your biggest critics into your biggest teachers. It's up to you to prove them wrong. Of course, if they get abusive, block them.

You can learn so much on the Internet, for the low, low price of your Ego. In fact, the concept of **Egoless Programming** extends as far back as 1971's The Psychology of Computer Programming. The first of its Ten Commandments is to **understand and accept that you will make mistakes**. There are plenty of other timeless takes on this idea, from Ego is a Distraction to Ego is the Enemy.

Don't try to *never* be wrong in public. This will only **slow** your pace of learning and output. A much better strategy is getting **really good at recovering from being wrong**. This allows you to *accelerate* the learning process because you no longer fear the downside!

1.4 Teach to Learn

"If you can't explain it simply, you don't understand it well enough." - *Albert Einstein*

Did I mention that teaching is the best way to Learn in Public? You only truly know something when you've tried teaching it to others. All at once you are forced to check your assumptions, introduce prerequisite concepts, structure content for completeness, and answer questions you never had.

Probably the most important skill in teaching is learning to **talk while you code**. It can be stressful but you can practice it like any other skill. It turns a mundane talk into a captivating high-wire act. It makes pair programming a joy rather than a chore. My best technical interviews have been where I ended up talking like I teach, instead of trying to prove myself. We're animals. We're attracted to confidence and can smell desperation.

1.5 Mentors, Mentees, and Becoming an Expert

Experts notice genuine learners. They'll want to help you. Don't tell them, but they just became your mentors. **This is so important I'm repeating it: Pick up what they put down**. Think of them as offering up quests for you to complete. When they say "Anyone willing to help with __ __?", you're that kid in the first row with your hand already raised. These are senior engineers, some of the most in-demand people in tech. They'll spend time with you, one-on-one, if you help them out (p.s. There's *always* something they need help on - by definition, they are too busy to do everything they want to do). You can't pay for this stuff. They'll teach you for free. Most people miss what's right in front of them. But not you.

"With so many junior devs out there, why will they help *me*?", you ask.

Because you Learn in Public. By teaching you they teach many. You amplify them. You have the one thing they don't: a beginner's mind. See how this works?

At some point, people will start asking *you* for help because of all the stuff you put out. 99% of developers are "dark" — they don't write or speak or participate in public tech discourse. But you do. You must be an expert, right? Your impostor syndrome will strike here, but ignore it. Answer as best as you can. When you're stuck or wrong, pass it up to your mentors.

Eventually, you will run out of mentors and will just have to keep solving problems on your own, based on your accumulated knowledge. You're still putting out content though. Notice the pattern?

Learn in Public.

> P.S. Eventually, they'll want to pay for your help too. A lot more than you'd expect.

1.6 Appendix: Why It Works

You might observe that I write more confidently here than anywhere else in the book. This confidence is based on two things:

- **Empirical foundation**: I have studied the careers of dozens of successful developers, and have personally heard from hundreds of others since I wrote the original essay.
- **Theoretical foundation**: Everything here is reinforced by well understood dynamics in human psychology and marketing.

We take advantage of these laws of human nature when we Learn in Public:

- The 1% Rule: "Only 1% of the users of a website add content, while the other 99% of the participants only lurk." You stand out simply by showing up.

- Cunningham's Law: "The best way to get the right answer on the Internet is not to ask a question; it's to post the wrong answer." Being publicly wrong *attracts* teachers, as long as you don't do it in such high quantity that people give up on you altogether. Conversely, **once you've gotten something wrong in public, you *never* forget it.**
- Positive Reinforcement: Building in a social feedback mechanism to your learning encourages more learning. As you build a track record and embark on more ambitious projects with implicit future promise, your public activity becomes a Commitment Device.
- Availability Bias: People confuse "first to mind" with "the best". But it doesn't matter — being "first to mind" on a topic means getting more questions, which gives the inputs needed to *become* the best. As Nathan Barry observed, Chris Coyier didn't start out as a CSS expert, but by writing CSS Tricks for a decade, he became one. **This bias is self-reinforcing because it is self-fulfilling.**
- Bloom's Taxonomy is an educational psychology model which describes modes of learning engagement — the lowest being **basic recall**. Learning in Public forces you toward the higher modes of learning, including **applying, analyzing, evaluating, and creating**.
- Inbound Marketing: Hubspot upended the marketing world by proving you didn't have to go out in front of people to sell. Instead, you can draw them to you by making clear who you are and what you do, offering valuable content upfront and leaning on the persuasive power of Reciprocity and Liking.

- Productizing Yourself: By creating learning exhaust, you can teach people and make friends in your sleep. This disconnects your networking, income, and general Luck Surface Area from your time. Don't end the week with Nothing. This is **Portable Personal Capital** that compounds over time and that you can take with you from company to company.

1.7 Appendix: Intellectual History

I didn't invent **Learn in Public**. The earliest mention I've found is this retrospective on how NASA Scientists do organizational Knowledge Management. Since then, everyone from Jeff Atwood to Kelsey Hightower to Kent C. Dodds attribute their success to some form of Learning in Public. Reid Hoffman, who studies great tech leaders from Brian Chesky to Jeff Weiner, calls it Explicit Learning. In fact, everyone you've ever heard of, dating back to Plato and Aristotle, you've heard of because they wrote down and shared what they thought they knew. Your learnings may outlive you.

I wasn't the first to benefit from this, and I won't be the last. The idea is now as much yours as it is mine. **Take it. Run with it. Go build an exceptional career in public!**