Experienced Developer Notes (Preview)

Steven E. Newton

2022-11-12T12:56:49

The last thing a programmer should be worried about is how fast they can type in code.

Everything is a people problem when you dig deep enough.

This is a preview version of *Experienced Developer Notes*

Personal Development Goals

The Five Whys

Ask 'why' five times? Yes, but do it by asking a lot of "what if" and "how" questions.

I Don't Know

If someone doesn't explain something to you and gets upset when you ask for an explanation, it's likely that person doesn't understand it what they're being asked to explain. With more experience and guidance comes the ability to say, "I don't know, let's try to figure it out together". But the hard part is getting over the fear of repercussions for saying "I don't know".

Friday Finish

Spend Fridays writing documentation. Complete whatever programming tasks for the week by Thursday. On Monday you have everything you did the week before written up.

Step Away from the Keyboard

There's a few things I can add to the other suggestions, but first, let's look at how we get to a point where we're stuck in work mode at quitting time. They thing to know is that you've got to start by arranging your day so you don't come to the end of the day fogged.

We can't do more than about four hours of "deep work" each day. Even getting a solid four hours takes practice. But fortunately our jobs are not all deep work. Yes, programming itself is deep work, but most of the workday is answering emails, following up on stuff, and dealing with toil. So start by planning your day to only expect four hours of focused, detailed work. Schedule time to do it at your peak productivity hours. For most people this means towards the beginning of the day. Do this work with frequent breaks. The pomodoro-style of 25 minutes on, 5 minutes off, continue for a couple of rounds then take a longer break. I use flow on my macbook to drive this. Make sure your rest breaks are real breaks. Get up, move around, stretch, get a snack. Don't just pop open a reddit tab and zone out reading. Disengage. I recommend Max Frenzel's work for further consideration. (Frenzel 2018).

Spend the rest of your work day on the other stuff, including meetings and such. Obviously most of us don't have 100% control of our schedules, but it's pretty common to see a senior developer block out hours a day to focus.

Once you've found a good balance of focus and rest, you should be able to get to the end of the day with a clearer mind. So now comes the transition routine. If you're working from home or remotely, you should have a space dedicated to work. Step away from the desk/computer. Change out of your work clothes. You did put on clothes in the morning, right? Don't just walk from the bed to work and spend the day in your pajamas. For me, this change of clothes, even if it's just my shirt and shoes, is what I call my "Mr. Rogers Time", from the TV show where Fred Rogers would always start and end by changing his shoes and sweater. Doing this gives you mental space to do something easy that helps your mind wind down. When I was commuting, the bus ride home was part of my transition routine.

Once you've put work behind you, mentally, and physically, you can do some meditation, exercise, yoga, or other hobby that's more physical than mental.

Tips For Code Janitors

Not available in this preview

Interviewing, Employment, and Work-Life

Money Isn't Everything

Some people are content living with modest means. Anything that tends to force the pool of candidates to act like they care about money above all else will have the effect of eliminating a large chunk of very good people from consideration.

Culture Fit

The thing about working at FAANG (and this holds true for the leading companies in other sectors, too) is that you have to "drink the kool-aid" and really buy into their way of thinking about problems and doing business. This is where the vague term "culture fit" really becomes clear: Employees have to believe that the company's own views about its success and goodness are true, and that critics and detractors are either wrong or have an axe to grind. Well, they either have to believe it or they are so focused on money that they have successfully compartmentalized any ethical or legal concerns out of their day-to-day work. There's also those desperate for a job any job, that they will work for a company they believe is unethical or full of bad actors, just to have a paycheck, but we software developers don't have to worry about that usually.

Coding Interviews Are Worthless

Let the companies that try to put candidates through the leetcode grinder know that no, developers, especially seniors, aren't going to put up with that garbage. It's a job seeker's market, let's set the conditions for employers to come to us.

When someone already working at FAANG, took 10 tries to pass an interview at another company? It shows the process isn't about figuring out if the candidate can do the job, it's selecting for candidates that can jump through the hoops.

Employers are definitely tilted against false positives, but that's simply because they have given up on actually developing employees. They'd rather roll the dice on a complicated and exhausting interview process in the belief they'll snag the perfect hire of someone who is a good culture fit and already knows the solutions to the employer's problems. There's no proof that these gantlet-style interviews are any good at determining someone's actual ability to do the job.

Companies don't want to hire great people. They want to not hire bad people. Nearly every management-oriented source out there giving advice on hiring puts "no false positives" (hiring someone who doesn't work out and must be fired) as the #1 goal. Employers don't want to bring in a person unless that person is from day one a productive fit for their organization. Employers don't do career development. Employers don't look past the current quarterly earnings report. Thus they see anyone who comes in and doesn't provide ROI in three months or less a net loss, even if that programmer, with proper coaching and mentoring, could be their future CTO.

I understand the temptation to want to find the perfect hire to slot into a skillset you're missing and code the solution, I had tried to do that once myself.

Of course, using a 6-hour marathon interview process as way to gate-keep so you only hire compliant people who will work 996 (9 a.m. to 9 p.m, six days a week) and never make waves is also a thing.

It used to be easier to vote with your feet in tech because transferring jobs could easily happen in a week. Current processes now make this significantly more difficult, on purpose.

Title Inflation

"Do not fall into the error of the artisan who boasts of twenty years experience in his craft while in fact he has had only one year of experience — twenty times." — Trevanian (Shibumi)

The job title "senior" doesn't mean anything consistently. The reasons are many but the main one from my experience is that it is the only way some employers will pay a market rate for decent programmers.

Title inflation is a thing. The other thing is that companies don't want to have real professional development programs. Instead of hiring entry-levels out of college and giving them entry-level things to do while actively helping them mature, companies always want to hire that "immediate impact" developer, and end up with a bunch folks who may be good but haven't developed in the company. Also, because there are no entry-levels to do the entry level thing, seniors end up having devote a significant chunk of their time to dealing with simple stuff. Sometimes that can be automated, and a good senior engineer always strives to automate those repetitive tasks. A better way would be to assign a new or junior engineer the task of automating some toil, with a real senior engineering mentoring, advising, and reviewing the work. Then the work gets done, the company gains a valuable experience employee AND the results of the work they did.

Someone getting a senior engineer title just because they hit their 5th (or worse, 3rd!) year in the industry but haven't really learned much in that time is just a flaw in the industry. Perhaps the meaning of "senior" differs across companies? There's a lot to know and just because someone has gotten really good at doing the things they've been doing for 3-5 years doesn't mean they've actually grown.

A novice senior is simply someone who has spent their career creating multiples of the same kinds of software using the same tools, languages, and processes as they did the first 2-3 years in the workforce, and can do nothing else. Those technologies may or may not be outdated and may or may not still be good ways of doing things, but the novice senior engineer knows no other way.

Organizations Look for Quick Fixes

No organization wants to spend money and invest in people they hire, they just want to find the unicorn that they can slot into a job and have them be productive from day one.

Mangled Job Descriptions

The ones that say <`num`> years of experience with <`language`> where <`num`> is greater than the length the time the language has existed or been used outside the group that created it. It sometimes happens that a hiring manager will want a person who has 10 years of experience, any experience, and happens to be experienced with a specific technology, and by the time the req gets laundered through HR and the recruiting pipeline the wording gets mangled until the job requirements are absurd and bear little resemblance to skills the hiring manager wants. Recruiters don't know that Java and JavaScript aren't the same thing. I wonder how many online job postings get loaded down with keywords for SEO?

What Does ‘Senior’ Mean?

Some employers expect senior engineers to be the most whiz-bang coders ever and not dirty their hands with ‘management-y’ stuff like quality initiatives. Others expect senior engineers to have advanced beyond just being able to code well to where they can see and understand the broader view and how the business actually makes money.

In my mind the first kind of company, the one that calls someone that can code well but isn’t able to work on the ‘soft’ stuff a senior engineer, is either just inflating titles, or they have a very traditional command-and-control structure with a bright line between IC and management.

In the other kind of company, a mid-level engineer that wants to be promoted should at least be starting to take on additional work beyond banging out tickets.

The typical job ladder for tech companies I’ve worked at shows increasing levels and breadth of impact the higher you go on the engineering ladder. A mid-level person might be expected to only do things that affect his immediate team, like adding a new tool to the team’s CI pipeline, a senior person’s work would have an affect across teams, encouraging adoption of tools, libraries, techniques, and processes that have a material affect on the bottom line.

I wouldn’t expect anyone looking for their first senior position to have *all* of these. But a senior engineer should at least be “in the hunt for” the skills listed below.

Years in the industry would be the last item on my list, because the other things I look for can’t be accomplished in a short time. Anyway, to jump into it, starting with Camille Fournier’s list I posted in another comment, I’ll summarize the key ideas as best I can.

Senior-level technical skills go beyond (and, depending on your perspective, may replace) being able to knock out the answer to a “hard” leetcode question. Some will say broad, others will say deep in a specific tech or stack. I say neither, but a little of both. The key is having some grasp of the entire tech stack from hardware to distributed systems. Not in-depth knowledge of everything, of course, that would be impossible. But I have yet to meet a good senior+ engineer that doesn’t have at least some familiarity with very low-level things like computer architecture and network physical and data link layers all the way up to concurrency, parallelism, and distributed networking systems. This should include some knowledge of compiler and OS concepts, application internal structuring and software design and deployment.

Communication skills land about second in importance. Not just good writing skills, but knowing how to structure design docs and specifications, how to communicate and explain technical things to non-technical people (without making them feel shamed or embarrassed) and pitching ideas to other engineers. While I don’t expect engineers to have the highest EQs, at the senior level they need to know how to shut up and listen, when and how to ask good questions, and how to make sure others know that you’re listening and understanding them.

Next I'd go with overall project technical and operational skills. Being able to set up a CI/CD pipeline from scratch to go from code to complete package or deployment. Build integration test setups that provide useful signals. Setting up monitoring and instrumentation for a running system.

After that, probably mentoring and developing less senior developers. Mentoring is a whole topic of its own that doesn't get enough attention, but even being able to notice a junior is struggling with something and say, "Hey I read/watched this book/article/video on that and I think you'd get something out of it for what you're working on." Ideally you'd work with them on it, not just drop it on them and go, but it depends on the you have and the organization.

Finally, a senior engineer needs to know their weaknesses as well as their strengths, and at least have some idea how to bring in another person to bolster their weaknesses. For myself, even though programming is detail-oriented, I'm surprisingly poor at a certain kind of tedious, repetitive detail work. I like to say I got into programming to be able to automate away having to do things like that, but sometimes you can't escape it. Realizing that has led me to understand that everyone has different abilities, and there's absolutely no excuse for disrespecting a co-worker who doesn't have your skills if their job doesn't require it. They almost certainly have skills you don't and quite possibly can knock out something with easy that I would be a struggle for me to even get a good start. Asking for help is not a weakness. Knowing when to ask for help and who to ask is a strength.

Why Not Coding Tests at Senior?

For mid-career to senior folks, why test them for coding? If they got that far they either know how to code well enough to do the job, or they bring enough value in other areas to make up for less-than-10x coding ability. Don't trust their resumé? If a candidate lies there, they have bigger problems than whether or not they can code, and it shouldn't be a challenge to screen them out. If the interview process can't screen out someone who lies on their resumé, a coding test won't save it.

The other thing about the typical coding tests popular now is that they don't actually determine if a candidate can be a good software developer, just that they can write solutions to artificial problems. Goodhart's law as stated by Marilyn Strathern: "When a measure becomes a target, it ceases to be a good measure."

Remember when questions like "How many golf balls can fit in a school bus?" or "Why are manhole covers round?" were popular? They were completely useless for determining if a person was any good at their job, but they were a nice way to gatekeep and ensure only people who happened to have the advantages that led them to knowing how to game the questions were able to pass the interview.

Rest of chapter available in the full version

Development Practices

Self-Organizing Teams

A good team should be self-organizing. Project managers are not the team wranglers, good developers should definitely not pass off teamwork as being the responsibility of someone else to oversee.

That individuals on the team have varying levels of organization skills is a given, but a high-performing team will have these aspect taken care of as part of the development process.

Good Naming Is the Essence of Abstraction

Entire discussions can be avoided by writing code in a way that doesn't depend on the person reading the code knowing the reasons for specifying how, but instead used concepts around the 'why'.

Technologies and Tools

Not available in this preview

References

- Bellotti, M. 2021. *Kill It with Fire: Manage Aging Computer Systems (and Future Proof Modern Ones)*. No Starch Press, Incorporated. https://nostarch.com/kill-it-fire.
- Bernstein, David Scott. 2015. *Beyond legacy code: nine practices to extend the life (and value) of your software*. Version P1.0 (August 2015). Place of publication not identified: Pragmatic Bookshelf.
- Binstock, Andrew. 2019. "Take Notes As You Code—Lots of 'Em!" November 21, 2019. https://blogs.oracle.com/javamagazine/take-notes-as-you-code-lots-of-em.
- Bussler, Christoph. 2020. "Online Database Migration by Dual-Write: This Is Not for Everyone." Google Cloud - Community (blog). August 27, 2020. <https://medium.com/google-cloud/online-database-migration-by-dual-write-this-is-not-for-everyone-cb4307118f4b.
- Cooper, Ian. 2017. "TDD, Where Did It All Go Wrong." December 20. https://www.youtube.com/watch?v=EZ05e7EMOLM.
- Crabill, Shannon. 2019. "Taking Note." Shannon Crabill — Front End Software Engineer (blog). July 15, 2019. https://shannoncrabill.com/blog/taking-notes-as-a-developer/.
- Feathers, Michael C. 2004. *Working effectively with legacy code*. Upper Saddle River, N.J: Prentice Hall PTR.
- Frenzel, Max. 2018. "In Praise of Deep Work, Full Disconnectivity and Deliberate Rest." January 9, 2018. https://maxfrenzel.medium.com/in-praise-of-deep-work-full-disconnectivity-and-deliberate-rest-e9fe5cc50a1d.
- Hochstein, Lorin. 2022. "Writing Docs Well: Why Should a Software Engineer Care?" Surfing Complexity (blog). November 24, 2022. https://surfingcomplexity.blog/2022/11/24/writing-docs-well-why-should-a-software-engineer-care/.
- Nygard, Michael T. 2007. *Release It! Design and Deploy Production-Ready Software*. The Pragmatic Programmers. Raleigh, N.C: Pragmatic Bookshelf.
- Spinellis, Diomidis. 2003. *Code Reading: The Open Source Perspective*. Effective Software Development Series. Boston, MA: Addison-Wesley.
- Thomas, David, and Andrew Hunt. 2020. *The pragmatic programmer your journey to mastery: 20th anniversary edition*.

Frenzel, Max. 2018. "In Praise of Deep Work, Full Disconnectivity and Deliberate Rest." January 9, 2018. https://maxfrenzel.medium.com/in-praise-of-deep-work-full-disconnectivity-and-deliberate-rest-e9fe5cc50a1d.