Daniel Ingraham

Dan has been around the block when it comes to coding (back-end to front-end in multiple languages), and is always happy to talk about a wide variety of topics, from nuclear physics to deer-composting. Basically, if you can think of it, he can chime in with some interesting tidbits about it!

Despite graduating with a degree in PoliSci, Dan jumped into the world of software engineering shortly after college. He's worked as a senior developer for several companies during his decade-plus software career, and is currently a senior engineer at InspectionXpert in Raleigh.

Email: daniel@danielingraham.com

GitHub: https://github.com/carrington

What's a change you've made in the past couple years that's helped you in work, life, or both?

Forcing myself to take half an hour between finishing a task and testing the task. I find far more defects when I'm not experiencing the end-of-task relief and euphoria.

What book would you give to someone starting off in software?

"Code Complete[25]" and "The Structure and Interpretation of Computer Programs[26]", the former because it will make them a better coworker, and the latter because it will take them a few years

[25] "Code Complete 2" by Steve McConnell. A popular collection of software construction techniques (variable-namin' to deciding when to extrapolate out a function), among other helpful things.

[26] "The Structure and Interpretation of Computer Programs" by Hal Abelson, Jerry Sussman, and Julie Sussman. SICP focuses on discovering general patterns for problem-solving, and building software systems that use those patterns (recursion, abstraction, etc.).

to fully appreciate the book, but once they do they will grasp programming in a way that they never have before.

How did you get your first software job, and would you recommend a similar path for someone looking to get into programming now?

I got my first programming job before the advent of code schools, and I would definitely recommend trying a school first, rather than taking my path, which boiled down to responding to every posting for a PHP/Web developer on Monster.com.

What do you look for in a teammate or when hiring a programmer for your team?

A strong desire to learn.

What's a red flag for you when considering a new co-worker?

A tendency to assign blame. Programming is hard; everyone has more to do than they should ideally have, and when two coding minds collide there are always problems. Bugs are rarely one person's fault.

How do you stay up-to-date on new languages and technologies?

When I had more time, I would obsessively refresh Hacker News[27]. I still read it, but most of the time I keep my finger on the pulse of things by checking what's generating the most questions on Stack Overflow[28].

What's something you've struggled with in your career?

Early on, the stigma of being self-taught was my biggest problem. Lately it's slowing down long enough to test a solution from all angles, as well as finding the time to do everything I need to do.

[27] https://news.ycombinator.com/ - A popular site with late-breaking tech news and happenings.

[28] https://stackoverflow.com/ - Hugely popular site for programming questions and answers. Many of your coding-related searches will end up here.

What advice would you give to an aspiring developer? What advice would you tell them to ignore?

Find the tool that is easiest for you to comprehend and try to implement everything in it at least once, even if you don't have to. Code Katas, design patterns, whatever.

Don't let anyone tell you to switch languages or stacks because it's not "serious" or "pure" enough; PHP served massive demand for years despite being a "bad" language.

Spend time understanding the fundamentals of programming; you may never have to know what the optimal data structure for a given problem is, but you'll be a better programmer for learning why it's optimal.

Lastly, ignore the voice in your head that tells you you're not a real or good enough programmer. We all have that voice (or, at least, every engineer I've spoken to about the matter does) and it's the biggest impediment to getting better.[29]

What's an unusual habit or superstition you have with programming?

If something looks like it's working right the first time you run it, it's horrifically broken somehow.

What's something that surprised you about programming?

There's vastly more art and intuition in programming than is popularly appreciated.

What do you like to listen to while coding?

Podcasts; they somehow occupy the part of my brain that has a tendency to distract me from a complicated task.

[29]This feeling is known as Imposter Syndrome, and is really common among software developers (I've felt it pretty often, too).

Robbie Allen

The first time I saw Robbie was at a Ruby meetup in Raleigh; he stood up at the end and mentioned that he had formed his own company, Automated Insights, and was looking for some Ruby developers. Shortly after that, I was on my way to Ai as a software engineer.

Luckily, one of Robbie's big focuses at Ai was on making it a place where people would want to come to work, and I'd have to say he did a pretty dang good job of that.

Robbie's since sold Ai, and is now the co-founder and CEO of another tech startup known as Startomatic, which aims to make it easier than ever for entrepreneurs to start their own company. He has over 20 years of experience in the software world, including programming, being a writer and editor for O'Reilly, and being a successful startup founder.

What book would you give to someone starting off in software?

Mythical Man Month[30]

How did you get your first software job, and would you recommend a similar path for someone looking to get into programming now?

I highly recommend college students do one or more internships.

What do you look for in a teammate or when hiring a programmer for your team?

Eager to learn, humble, positive, and smart.

[30] "The Mythical Man-Month: Essays on Software Engineering" by Fred Brooks, who helped found the computer science department at UNC. A book about software development and project management that drives home the point that "adding manpower to a late software project makes it later" (known as Brooks' law).