JARED: Today we're talking to Dave Gauer of ratfucker.com.

DAVE: Ooh, Jared, I think you might have been on the wrong website.

JARED: Welcome to Dead Code. I'm Jared. Today we are talking about, well, we're talking about a lot of things. This is probably our longest episode in recent memory. We are going to be talking about Zig, the programming language, Ziglings, the exercise-based tutorial for learning Zig. And then, we'll be talking about why and how you should blog the Forth programming language, if you're familiar with that, and just a lot of other things along the way. We let this one meander because there was just a lot of fun, interesting stuff. So, let's get into it.

Dave, welcome to the podcast.

DAVE: Thanks for having me.

JARED: So, could you introduce yourself to the audience?

DAVE: Yeah, I'm Dave Gauer, and I have a website. I've [laughs] been making webpages since the mid-1990s. I've been in the industry since 1999. I've been a web developer the whole time, and I have a ridiculous range of hobbies outside of work. But online, I would say that my focus, especially increasingly, is working in public and learning in public. I really like to work on small, fun software projects, things that just tickle my fancy at the moment. But maybe more importantly even than working on those projects is writing about them and documenting them, and bringing people along for the ride, if you will.

And I'm probably best known for Ziglings, the learning project, and also a really long article about the Forth programming language, which had [laughs] got some pretty good traction. And, otherwise, I post art on Mastodon. And, I mean, that's largely my online presence.

JARED: So, for our audience that might not be familiar with Ziglings, what is Ziglings?

DAVE: Yeah, so, I don't even know to what degree...I think Zig is pretty popular now that that's...it's pretty well known as a language. But I first heard about it in 2016 when Andrew Kelley had his initial post, and it's, like, hey, this is a replacement for C, and we're going to specifically target that. A lot of people try to avoid saying, "Yes, we're going to replace the big dog," and, yeah, he just came right out and said it. So, I thought that was cool, but I didn't really need that.

So, it wasn't until 2019...he made a presentation or a talk called "The Road to Zig 1.0," which is just, I think, a master class in explaining what this thing is for and why you would want it and essentially explaining that here's a systems-level programming language and framing it in terms of, well, here's C and all the things we know that are wrong about it, especially, like, what's it called, the precompiler, the include statements and such.

JARED: Oh, like the macros? Yeah.

DAVE: Yeah. So [laughs], C's ifs and endifs and this really crufty stuff, the limitations of that language, how can you just take those and improve them? A lot of it was framed as why not? Anyway, I'm probably talking too much about Zig itself, but that really got me interested. I'm like, this is a really well-thought-out project. That's what impressed me the most.

And so, I was actually just looking at the commit log for Zigling. So, it's December 23rd, 2020. I'm super excited about this. It's got my shiny stuff senses going, and so I'm on holiday break. And I had just done Rustlings, which is this really clever series of exercises for Rust. And what I thought was really great about this is that it dumped you right into a broken program that you had to fix. And in fixing that program, you weren't just reading documentation. You were interacting with the compiler immediately, getting real errors. And [chuckles] as you fixed it, it gave you this nice feedback. So, it was very sort of game progress in feeling.

And so, I thought, yeah, let's try doing that for Zig. Why not? And so, I wrote a bash script that would sort of run this whole thing. And I did my first "Hello, World" initial program that you had to fix, and it just went from there. And then, it looks like about a month and a half later, I'd written something like 40 exercises and gotten through some percentage of the basic language. And I thought, yeah, I think maybe I can sort of turn this loose on the public and see what happens.

And the thing to keep in mind is that I didn't know Zig. This was a process of me learning the language and immediately teaching it. So, literally every file that you see in this thing that is an exercise is also my first attempt at learning that feature. So, you want to talk about learning in public; this is it. And, in fact, that was about to ramp up because I did...I think I posted this to Reddit initially, and I was really surprised at how popular it was, I think because beginner stuff is popular.

And I think also because I put a kind of a cute spin on it. I've got this drawing on the README. So, I used the, what is it, the Ziguana and Zero is the Zig mascot. [chuckles] And I tried to make it as cute as possible. They're working on little broken gadgets or what have you. So, the idea is that you're in there fixing things. You're fixing little programs. That's the tagline I added. So, the idea is that you're never confronted with...actually, that's not true. I'm about to say something not true.

JARED: [laughs]

DAVE: You're never confronted with a huge, big problem. But actually, I had to change the README later. People were really upset. There is one large exercise in this that really throws people off. But I'm warning you now in the README that there's one nasty surprise, and it is a much larger exercise. And I think, in a way, that one is...if you were to rate by controversial, this exercise would rise immediately to the top because a lot of people hate it. Intriguingly, a lot of people love it. It's their favorite, I think, because it does dump you into something a little bigger, and it forces you to deal with a larger chunk of code, which is a little more realistic.

JARED: Do you think it's important that you include that sort of outlier in terms of the complexity of these exercises?

DAVE: I debated it, and I still debate that to this day. And, in fact, I think I made that an open question when I still had this on GitHub to ask the community at large, “Should I just remove this thing because, you know, it is upsetting people [laughs]?” But, yeah, ultimately...well, I didn't get a clear answer, and the truth is I still don't know if I should even still have that exercise in there. People seem to get through it.

And I did go back in, and I think I did smooth things out a little bit, made it a little clearer, and a little more hand-holding, as well as some encouragement, just literally writing out, hey, this is much bigger than the previous exercises, just making that very clear. You know, we're not just dumping you into a lake now, and you're going to die. It's, hey, just take it slow. It'll be fine [laughs]. And that does seem to have worked, and I haven't seen as many complaints after that. So, I think people just needed to know that they would survive this.

But, yeah, it has been an incredibly popular thing. What's funny about this...so after I published it, I continued to add exercises. It's up over 100 now.

JARED: Oh.

DAVE: I wrote the initial first 90, I want to say, and after about 40, I am literally learning in public right in front of people that actually know this language, and [laughs], you know, so no pressure. And, you know, I think just like the cute, little mascot characters get you in there and sort of set you at ease, I think I get away with not necessarily knowing what I'm doing while I'm building this because I've got a sort of a whimsical take on the whole thing.

So, if you read through Ziglings all the way through, the first ones are pretty run-of-the-mill. It's “Hello, World,” and I really felt like I couldn't be too cute to begin with. I didn't want to throw people off and confuse people. I'm also...I try to be really aware of people who may not speak English as a first language, so I try to be really careful about wording things as simply as possible in the beginning.

But as you go, they get sillier and sillier, and I actually think they're kind of fun. And I say that even though I wrote the bulk of these because I actually ended up going through this essentially as a newbie later to relearn Zig. So, in 2023, I'm having a heck of a time keeping up with this thing. So, I should back up a bit and say that one of the things that is unique about Ziglings versus any documentation you might find, et cetera, other than the official documentation, which is always up to date, is that this tracks the master branch of Zig.

And so, pretty much at any given time, now, it could literally break overnight, but at any given time, if you grab the master branch of Zig, you can use Ziglings with that. So, we don't do anything special for major version releases or what have you. And that's largely because the

language, especially at that time, was changing so fast that to get stuck at any particular point in time was you were really being done a disservice. You were learning stuff that was literally not true, maybe a month later. So, that was very much a conscious choice.

And because of this, because it is current, that has baked into it a certain amount of responsibility to keep it up to date. And at that particular time in my life...so, I've got kids. I've got a full-time job, you know, things happen. I was not able to do this anymore. And so, near the end of 2022, give or take, I had a contributor who was helping a ton, in fact, to the point where he was doing more on this than I was. And I finally said, “Hey, you know what, would you like to be a co-contributor on this?” And I meant it at the time.

What I didn't realize is that I was really just offering for him to take this over [chuckles] entirely. I don't even know if he would have taken it if that had been what I had offered. And I certainly didn't realize how little I would be involved with it later. But his name is Chris Boesch, and he's in Germany. And he has been incredible at keeping up with the community requests and PRs and responding to comments.

And what's really incredible about this, to me, is it really is an open-source success story insofar as I have been able to just step out of the picture. I was able to leave entirely, stop being the bottleneck. I check in occasionally [chuckles] to see how it's doing. But I love that this is out there just living on its own, and that I get to say that I started it, but I don't have to keep up with it. And that's awesome. That is a really cool feeling. I've never had a project like this before. I've worked for companies where things live beyond just you as the developer, but I've never had a personal project that lived beyond myself.

JARED: Yeah, it's very cool. So, Ziglings explicitly states that it's for people who have never done systems programming before. How did you design those exercises to sort of bridge that gap between higher-level programming experience and then, you know, this relatively, you know, the C replacement language that is, you know, from most people's experience, a lower level?

DAVE: Yeah, that's a great question. If anything, it's just what I didn't do, which is anytime I encountered something that would be common knowledge for a C developer, I didn't let that just be common knowledge and go without explaining. So, I really did try to explain everything as tersely as possible and with as good a humor as possible, but without leaving any gaps, to the best of my ability.

Of course, all of us are very much unaware of what we know and what we take for granted. I will say, approaching this as a non-C developer was very easy for me because I am not a C developer [laughter]. And so, in fact, the things that I didn't know are also the things that other non-systems programming language experts would also not know, and so that gave me a real leg up on understanding that.

But, to be sure, even things that I do know, I made sure to make explicit, and most of those are in the form of comments. There are comments in the exercise. Some of them are at the top, which tends to be more explanation about what you're about to do. So, I'll say, you know, you've got...here's this new language feature, and here's how you might use it. And then, there'll be an exercise that is broken for one reason or another, and knowing how that feature should work helps you fix that, so you're good to go.

But below that, there will be additional comments, which might give either some extra flavor, so you'll just learn some tangential things, because I can't help it, because I think they're interesting, so maybe somebody else will. Or they will further explain something that somebody may not know going into this world of bits and bytes, for those of us who are used to dynamic languages, and really just dealing with strings and never having to manage memory and things like that.

JARED: So, one of the things that sort of piqued my interest in Ziglings is I have been sort of toying with the idea of building an interactive Ruby tutorial. I'm a Rubyist myself, and Ruby has a sort of history of a lot of whimsy around the language, as well, with “Why’s (Poignant) Guide” being probably the most famous sort of example of that really taken to an extreme. What role does narrative play in this kind of technical education, and how do you balance the whimsy with the learning objectives?

DAVE: Yeah, that really gets to the heart of, I think, what I both enjoy and what I think is important, and I think what's compelling to people. So, yes, the whimsy especially...yeah, I'm a fan of “Why’s (Poignant) Guide.” I wish I could buy a paper copy of that, by the way, because...

JARED: You can, I think, now, again.

DAVE: Is it? Okay.

JARED: Somebody has been printing...I have one. I have, like, a new...from a few years ago. I have a new copy of it. So, I think...I don't remember who it is, but somebody's printing them.

DAVE: Okay, nice, because I love books. That's a big thing for me in general, and, in fact, I grabbed a couple off my shelf. This is in no way a top ten or anything, but some of the books with whimsy, for example, I've got “Learning Perl,” which is one of my first really pleasurable experiences reading a programming book. It's kind of just funny right from the outset. A lot of it's dad-joke level stuff.

JARED: Yeah [chuckles].

DAVE: But I think it really puts you at ease, like, hearing the preface: “What is this book about? Among other things, this book is about 260 pages long [laughter].” Opinions will vary on how good that joke is, but it gives you the right idea.

JARED: Yeah, it sets the tone.

DAVE: Yeah, I really enjoyed that, especially when I was first starting, because everything else was so intimidating.

JARED: Yeah, and dry, usually.

DAVE: And so dry, yes. And that is something that sets the Ruby community apart, and Perl as well was really known for that.

The other one that...and, I think, maybe I reference this in the Zigling's README is “The Little Schemer.” I believe its predecessor was “The Little LISper,” but either way, it's by Daniel P. Friedman, and this book is really something. It starts off with [laughs]...the first exercise, you're not typing on a computer at all. It's literally a question: "Is it true that this is an atom?" And then, it's the word atom, and the answer is yes, and then they explain why. And then, the next question is, "Is it true that this is an atom?" And then it's the word Turkey [chuckles], and that is also an atom.

So, it's really silly. And it also has these really cute drawings of little elephants done in this pen and ink style that I love. So, the balance of, yes, that's silly, but also, it's so bite-sized that the silliness is not getting in the way of learning something. If there's any criticism to the Poignant Guide, it's just that, you know, finding the programming tidbits amongst [laughs] all of the fun stuff can be a little challenging. So, maybe that's not the book that you necessarily hand to everybody to learn Ruby, although what a world it would be if we did. That would be a kinder [laughs], gentler world, wouldn't it?

JARED: I think so.

DAVE: But, yes, part of the reason I think that the whimsy is important is to take some of the scary out of it, take some of the seriousness out of the subject, and to make it feel more approachable for everyone. I think that, increasingly, that's my reason for getting online these days is to try to just make people feel like they can do this stuff.

I think as we get increasingly complicated, our stacks get taller and taller. I think it's getting...I can't even imagine starting right now, honestly, so, you know, like I said, I go back to the ‘90s. We learned how to make web pages by right-clicking in View Source, and you copied the parts you liked into a text file, and you saved it to see what happened, and you refreshed. And that simplicity, just you and a text editor, has enormous appeal for me.

Big build systems and things like this, I've seen other people complain about this. But you look at how to make a web page, and the guide is nuts. They'll be, like, well, you install NPM, and you've [laughs] got this enormous list of build tools that you need to install. And I'm, like, wow, that's nuts because I still don't do that stuff. I'm a professional. I'm still in the web dev world. And, yes, we use those tools at work. But at home, I don't use that stuff at all, none of it. I sit

down with a text editor, just like it was the ‘90s, and it's me and some HTML, or it's me and some PHP, some JavaScript, some CSS, just the basics. And I'm making little tools.

So, I think approaching these things as, well, first off, just explaining that they are approachable on their own, that we can learn these things as their own little technologies. You don't have to take all this on at once. Web dev is nuts because we have this accumulated craft of decades now of adding, never removing, always adding, adding, adding. And so, this stack is enormous, and you're also expected to learn an enormous number of tools just right off the bat. And, well, I think that's wrong. That's me swimming upstream.

JARED: Yeah [chuckles].

DAVE: I can't change everybody else. But at least on the things I do, I can, I hope, show people that, yeah, actually, you still can do these really simple things and make something that's compelling, and interesting, and it actually works. And approaching, say, Zig, for example, as this little self-contained environment that both teaches you, and you're learning it, and you're interacting with the real tool, and you're using your text editor, just like you really will when you leave Ziglings and start writing software on your own. Those tools are not going to change. It's the exact same stuff, and yet it's still a self-contained environment.

You know, I've got another author who I think is really interesting, unfortunately, no longer with us, but Seymour Papert who his book, “Mindstorms”, is what I'm holding in my hand right now, I found quite influential. His deal was teaching children to learn, well, computers, but also math, and also just mathematical thinking, or how to program, how to think logically. And he's best known for the language Logo and the Turtle, the little turtle that you program to go left and right, and forward.

So, this book, “Mindstorms,” I'm pretty sure it's from this book, or maybe something I read from him later. He's got this concept of microworlds, and I've written about this in scattered places on my website. I keep coming back to this. I really like the idea of these self-contained, I'm just going to use his term, these self-contained microworlds wherein everything you need to know to learn it is there. You don't have to go to any external resource to look anything up. And so, if you were, you know, dropped on an island with a laptop with this loaded, you've got everything you need. Just time and patience, and this thing, and you will learn how to use it, all in this one place. And that is so appealing to me.

I think just the idea of a self-contained world, in general, is appealing, you know, like a little puzzle box that you could just sit on the couch with [laughs] is appealing. And, you know, kind of as a side tangent, I recently wrote a little page about tiny programs as well, small things, and why we find those appealing. And, at first, I'm talking about small programs that, you know, do things, and why we find those appealing. And I think it's because the small size means that you know you can learn it. It's just, you know, you take, like, the codebase for a browser or an operating system. You're not confident that if dropped on an island, you could ever learn that. There is just so much.

A small enough program, even if it's a ray tracer or something that seems conventionally difficult to understand, if it fits on a business card, suddenly it's like, yeah, I think, given enough time [chuckles], surely, I could understand this. There's only, like, a thousand characters here. How could I not understand this? And so, that takes some of the intimidation out of otherwise really difficult-to-understand things.

And later on in that article, I had found a couple of other writings that people had done about tiny things: why we like small things like dollhouses, or why we like just little toys or toy models, and things like that. And it ultimately comes down to, yeah, having control over a thing, even if the rest of the world is this chaotic thing that you'll never get a handle on. We will never understand all of web development as it currently stands.

You know, for example, I've never written a single line of Ruby on Rails. I actually use Ruby a lot, by the way, but it is essentially in the original intent of a command-line program. I really like it for that purpose. But Rails is a whole universe entirely of its own that I don't know at all. And it's kind of amazing that I've been doing this professionally for 20 whatever years, and I've never even touched it. It's a whole universe. I've never even touched it.

So, we know that we can't do everything, and we know that's okay. But knowing that there are things that we can learn literally everything about is so appealing to me. I think that's also maybe, again, another tangent, but I think maybe that's part of the appeal of retrocomputing. You have these old computers that were so simple that you could understand everything about it.

You know, you look back at the old machines, like advertisements for old computers, old home computers. They came with manuals that covered the whole machine, hardware to software as a whole. And you absolutely could sit there in your living room, or wherever this thing ended up being set up, and learn the entire machine inside out. And I think that probably inspired a lot of people that are big names now because they found the appeal in that. And while we're never going to go back to the 1980s --

[chuckles]

JARED: Yeah. I've seen the current Intel manuals.

DAVE: [laughs]

JARED: You can get copies of them, but they'll take up your whole bookshelf.

DAVE: It is incredible, yeah. And that's just the processing. And you're not going to understand that. You're not going to understand all of that. You know, I take that back. If you landed on an island with just those manuals, I think you could take that [laughs] in --

JARED: Yeah, you'd get there eventually

DAVE: Yeah, but at what cost [laughter]? So, yeah, if you take this idea of the self-contained world and present that to people, I think that's very appealing. And so, as a learning resource, having something that essentially comforts people by saying, "Look, if you do this, if you stick with this little thing, you're not going to need to go elsewhere. You're not going to need to buy anything. It's all right here, and you will learn this," it's essentially like I've given you a video game, and just stick with it. It will teach you itself.

I've got another sort of, I don't even know, I guess I'll call it a project, but it's, yeah, how would you call it? A state of mind, almost. So, I've got this series of webpages. It's called Assembly Nights. And what I did, I don't know where I got the idea, honestly, but I had this little netbook PC. It's the original Asus Eee PC.

JARED: Awesome.

DAVE: And this thing can, what's so great about it, I mean, it does have a...it has Wi-Fi, but generally speaking, this thing is not going to go on the modern web. It is not capable of that. And already you've got a distraction-free device. And so, I've installed many different OSs on this little Eee PC over the years, but I ended up with Alpine Linux because it's a very lightweight, sort of BSD-like Linux. And I started programming in Assembly on this computer, and I was doing this at night as well. And so, it was this weird, little...I like to read when I go to bed, typically. And I've always done that ever since childhood. It calms my brain down. Otherwise, I lay there and I think, and I think.

This, strangely, you would think that this was a really, really bad idea for somebody like me to go to bed with a laptop. But because this was a little self-contained machine that could barely go on the net, I actually did use Lynx, the terminal browser –-

JARED: [chuckles] Nice.

DAVE: To [laughs] read some docs occasionally. But I ended up porting a Forth as my project. And I ported it from one flavor of Assembly to another; that was my project. And so, I ended up learning three things there. I learned one flavor of Assembly, another flavor of Assembly, and Forth, all at the same time. And every night, I was in this little world, sort of like you're in another world when you're reading a novel. And so, it was actually very relaxing, very, very different from our typical day-to-day programming where we're constantly looking up references, battling third-party APIs, whatever --

JARED: Yeah. Living in a much larger world.

DAVE: Yes, trying to understand what other developers have written. Yeah, a much larger world, exactly. This was a smaller world, a much smaller world. And I think I wrote something to the

effect of, I'm here on this tiny, little computer writing tiny, little instructions, and [laughs] that is really true. And so, it ended up being one of the projects I had stuck with for the longest time.

I think I ended up taking...I'm just going to go off the top of my head, but let's say it was a year that I stuck with this. And maybe that doesn't sound like a lot, but you'll have to trust me when I say that that is extremely [laughs] unusual for me. At least up until that point, I have always been the type that will chase new, shiny projects. And so, finishing things has always been a very big challenge for me.

And this, for some reason, I was able to stick with. And I think it's because it was genuinely pleasurable to go into this little world. And I think also because it had no distractions. So, while I was doing this, I was probably tempted to look up, like, “Oh, what's the best way to do such and such?” and then start reading blog posts [laughs].

JARED: Been there.

DAVE: Oh man. Yeah. That's my day-to-day if I'm not careful. So, I think it was that combination of distraction-free, plus, yeah, just the joy of being in this self-contained thing that I had complete control over that kept me with it.

JARED: For our audience, you mentioned Forth earlier as well, but could you explain what Forth is?

DAVE: Oh, gosh, I would love to. So, this guy, Charles Moore, has been assigned a task at, I believe, a university at the time, and he is writing in Assembly because everybody is. You know, what's great is I've got this article with all the machines that he used as he was going. And, basically, as he went, he started to evolve this system that was just his tooling to help him build a system that was flexible.

In a lot of ways, Forth, really [chuckles], I would argue, started as configuration for Assembly to sort of tie together some Assembly that he'd written. And by the time you get to...it may or may not have been the fourth computer he was on, but it was the fourth generation of this system that he had been working on. He went to give it a name, and the file name convention required that he drop some letters. So, rather than fourth with a U, the number four, he ended up calling it Forth. So, now we have a programming language called Forth. But really, this thing it's like a box of gears; it's more like a watch with a series of complications. And it works. And it's kind of amazing that it works because it is a series of really simple pieces.

What I love about it is what I've just described does not sound like a programming language. And it's a little bit intentional because I think if you approach Forth as just a normal programming language, which you can, I think you're missing what this thing...how it started, and therefore why it exists. But it is a programming language, in fact, and it does even have a standard, and a standard library that you're expected to have, etcetera.

Forth is, if you want to be fancy, you would call it one of a category called the concatenative programming languages. Often, these are called stack-based, and those two terms are kind of interchangeable. I like concatenative because that gets more to the heart of sort of the after-the-fact theory of Forth, which is that you are working with this tacit data that you keep putting onto a stack. And as you compute with that data, at no point are you naming that. You can. You can create variables and things, but generally speaking, you're just working with verbs just strung together like a sentence that happen to work with data that sort of gets accreted from one verb to the next. And ultimately, you end up with something that looks very human language-like in a lot of weird ways.

It is both one of the lowest-level languages anybody will ever touch. It is very close to Assembly in terms of its ability to, I mean, you're literally dealing with, at its lowest level, jumping instructions and things, so all the way down to that. In some ways, it is a much higher level than many other languages because you're building words. They go in your dictionary, and this becomes your own language. So, you're really encouraged to build a DSL as you're programming in Forth, and that's pretty unusual.

I know that Lisps and others also kind of have this do-it-yourself reputation, but I think Forth really takes that to another level. It is also sometimes considered to be write-only [chuckles], like many languages are for that reason, that if you were to walk into somebody else's Forth codebase, you might be either amused and delighted or shocked and horrified by what you see because, at least if done correctly, I think if you're following Chuck Moore's philosophy, it will be your language, truly.

I think it's one of those where a company culture would develop around this dictionary that you've built up that would be truly unique in a way that just standard coding conventions don't really quite encapsulate, you know, normie language [laughs] companies. And Forth is still used to this day. It's got its niches, like many of these old languages. It's not quite like COBOL, where it still exists just because there's so much of it that we can't just easily replace it. It exists in its niches where people actually like it, and I think that's because of this just ridiculous complexity.

In the course of this enormous article I wrote, it's something like 30,000 words, and I did, I think, 70 drawings in this. I teased this thing for Forth, which is a legend, essentially, that I had heard when I first heard about it, which is that you can change the value of numbers [laughs].

JARED: Yes. Yeah. Something you can't even do in Ruby.

DAVE: Right. Exactly. Yeah. So, for all the maligning that Ruby gets for its [laughs] extreme flexibility, yeah, Forth really takes that to another level, and that's one of the first things I tried out. Once I learned enough Forth, I tried it out, and it is true. And so, by the end of the article, spoiler alert: yes, you can. And so, I set the value of the number 4 to 12, and sure enough, if you type 4 in that running copy of Forth from then on out, you're getting 12, which really points to something that would horrify, I think, a lot of developers.

And I know this is something that horrifies devs in even just medium-sized companies about, yeah, say, Ruby or JavaScript. Anything where we don't have strong typing, you can kind of just make stuff up as you go. I love those things, personally, maybe not for large codebases; it can be troublesome, but for personal programming, that's just fun.

Yeah, so Forth, like I said, it's got its niches. Another niche where Forth shows up, I don't know if this is still true, but some pretty recent instances of this are true. Forth had a place in space. So various technologies that have made it to space run on chips that run Forth natively. They actually have two stacks built into the CPU, and you can read about those.

I've got a nascent page up in the Forth section of my website called Forth in Space, and I'm doing my best to kind of fill that in slowly with what I have gathered. Most of it's based on a NASA document that somebody at NASA started. What you find out is that a lot of them end up being a chip that is radiation-hardened, largely due to the way it was manufactured, and so it's just a natural fit. But they're extremely low-latency, and Forth really lends itself to this sort of low-latency programming as well.

But the really interesting thing about that chip, specifically, is that it is a design that was initially created by Chuck Moore. So, it turns out that not only did he do this really, really innovative programming that nobody else really had done anything quite like that before, but also the guy is a really interesting hardware creator as well. In fact, there's a, oh gosh, is it called GreenArrays? I believe I have that name correct. He has a talk that you can find on YouTube where he shows the GreenArrays chip that has, what is it, like 140 cores.

JARED: Wow.

DAVE: And each of those is running this stripped-down ColorForth that allows you to talk to the left and right cores, so you're talking to neighbor cores. Each one is individually programmed. But how you would use this effectively, I have no idea. But I find that talk compelling for a whole number of reasons. The power usage, also, he talks about chips that could maybe just run off of the physical motion of someone wearing them, so incredibly low power usage, which I also find super appealing.

I think little systems that could run off of solar power and things like that are where maybe Forth could find another life. And I find that really appealing, too, that something so old cannot only continue to exist, but might be pointing at a future that we could have, too, if we have lots of little systems that are really, really low electrical usage.

JARED: Yeah, Forth has been a language that I've heard about so many times and never taken a serious look at. And every time I learn a little bit more about it, I continue to be even more intrigued. So, I'm going to have to sit down and check it out at some point here.

Bringing it back to Ziglings, what do you think is the most surprising piece of feedback you've received around Ziglings or from Ziglings users?

DAVE: The feedback that we mentioned about this large exercise, I think the fact that some people did like it was the surprising part. The other thing that was surprising was just how popular it was. I think that's probably still, by far, the most popular thing I've ever done. And while I've written quite a few tutorial-type things, this really seems to appeal, I think, to younger programmers as well.

So, I come from a learn from a book generation just because that's what we had. That is all we had. But, yeah, I think I was a little blown away by how popular it continues to be and the age range. Now, mind you, I'm definitely generalizing based on the avatar images that people use to try to guess how old [laughs] they might be or just the language they use. And I could be way off, but it seems like a younger audience. I think a younger audience is attracted to Zig for various reasons as well. But this, in particular, I think, the immediate interactive nature is appealing to a younger audience.

I think what's funny about Ziglings is that I used it both as something that was hopefully useful for others, but also as a learning resource for myself, just to learn this initially, to give me a structure to learn Zig. But I don't really learn this way. I actually learn better from [laughs] a book, doing my own exercises that I make up myself. So, while I enjoyed Rustlings, I did eventually just go to the Rust manual, which was incredibly well written, and learn it traditionally as well, or what I would call traditionally, although increasingly, that's sort of an oddball thing to do, I think [laughs], to hit the books.

But yeah, I think the age of the audience has been surprising. And so, while I've not gotten direct feedback from those folks, I've seen chatter about it. There's a Ziglings Discord that I'll occasionally check in, not Ziglings, sorry, a Zig Discord that I will check in on occasionally. And so, I occasionally see mentions of Ziglings, which, of course, gives me a huge smile on my face when I see that.

And it's really cool to see people who are clearly beginners because they're struggling to get Zig going in general, but seem to do all right with this and enjoy it and have fun with it. And, I don't know, it kind of blows me away that something that I intended actually worked out the way I intended it [laughter]. I've had successes before, but not because I intended them to be [laughs]. So, it is so neat to see that it's doing exactly what I wanted it to do. So yeah, that's been really neat. And it's all been positive. All of the feedback has been really positive.

I think I've also been maybe not surprised but pleased to see that people have enjoyed the whimsical aspect as well. And I think maybe this is a thing...I think this is a thing that is increasing, but well, I'll just say it, less tech bro vibes on programming in general. So, less of a real, I don't know, I've got to be this tough cyberpunk in a leather [laughter] jacket with the sunglasses to be a real programmer feel.

I've seen a lot more use of color on web pages explaining programming topics, things like that. It's not just the white text on black screen, although I love those websites, too, don't get me

wrong. And, in fact, those websites are the ones that taught me so much of what I know. And I am truly grateful for the people that made those sites, those people volunteering to put that information out there. They had no reason to do that, maybe just any fame that would come along [laughs] with explaining something well.

But those people really, and I'm talking, like, the late ‘90s, the people that made some of the first free instructional stuff on the web with no expectation whatsoever that anybody would maybe even necessarily read it, did so much for the rest of us. And I don't know if they'll ever really know how much, but that was hugely formative for me starting out. And I hope that I'm doing my little part to continue that, that grand tradition of, yeah, just, you could call it pay it forward if you want, but adding to this collective knowledge that we all have and doing it in a way that is hopefully fun and inviting.

And, yeah, in a tech world that can be pretty off-putting in many respects, yeah, making something that feels personal and fun is important to me [laughs]. And I think it will attract the right kind of people and the right kind of thinking as well. And, yeah, I think if anything, I hope that by being sort of established in this industry and then doing silly stuff, I hope that I'm kind of making it, I don't know, accepted or giving permission to others to do funny stuff, that we don't have to be serious even if we're, you know, [vocalization] I've been doing this for so long [laughs]. But, yeah, that we can still have fun because that is really, really important to me. I gather that that's important for you as well, that you've got on.

JARED: Yes, yeah. I got into programming to have fun, to make games when I was a kid, and learned a little bit about what the games industry was actually like, and pivoted away from that. But still, I'm here for the fun and the wonder and the joy of programming. Well, I'm not sure what other thing I would be here for, but whatever it is, that's not what I'm here for.

DAVE: Yeah, exactly. I do think that we're lucky to be able to pay the bills doing this thing. I feel incredibly fortunate. I mean, I really came into this at a truly golden time when I didn't need any experience or a degree or anything, and I wish everybody could have that. I truly do. You know, even just right as the dot-com bubble was bursting, even some part of my young, stupid heart knew that that was going to be a lot harder for others to follow the same easy path that I had, and I hated that idea. So, anything that helps level that playing field past me, because would I be able to have entered now and do this stuff? I don't know.

JARED: Yeah, it's something I tell people when they ask me for advice on how to get into the industry is I have absolutely no idea. The path that I took is not there anymore.

DAVE: I see the same thing from published authors as well, you know, the traditional path into publishing houses is just no more. They have no idea how you do this. I think, also, the need to be publicly available is different now for all of us, programmers as well, to make a name for yourself. Having a presence outside of a particular company so that you continue to exist with your accomplishments after you leave a place can be really challenging, especially if you're just

writing code all day that is private property. So, I think, yeah, establishing yourself. I don't know how you would do that now, other than just doing the work.

Well, I actually do have kind of an answer to that. So, there's this book. I've got the sequel right here. It's driving me nuts that I cannot find the first book. So, this author, Austin Kleon is his name, has a book called “Steal Like an Artist.” Actually, I hate that title [laughs], but he explains the concept very well. But in it, he's got a bunch of really great chapter headings. The whole book is...and it sounds like you're familiar with this book.

JARED: I've heard of it, yeah.

DAVE: Yeah. I highly recommend this for everyone, whether you're an artist or not. And, I mean, honestly, we're all artists to some degree, aren't we? Especially us developers, I think a case could be made for that. Anyway, I think this stuff's really pertinent.

So, Chapter 2 is, "Don't wait until you know who you are to get started." And I really like that because...so, I've got this website, ratfactor.com, that I've had...I had just left high school, and I get a domain name. It was really expensive to get a domain name. I could have had English words, like common English nouns as a domain name [laughs] when I got this. I actually don't regret it in the long run, having a weird name because it's unique. So, nobody else has taken this.

But I had no idea what to put on this website, other than just a few things that I had already made. And for the longest time, I struggled with what to put on this. You know, is it a resume? Should I just be keeping this work relevant and make myself look like a professional, especially starting out? Should I make this look like I should be hired for something, or should this be my art? Should I make this fun and have games there and things like that? And how do I balance those?

And I think the important thing is just to do something and keep doing something, and you will eventually figure out what that thing is. I really like...Chapter 3 is "Write the book you want to read." That is definitely the spirit behind something like Ziglings, where I like this whimsical stuff. I'm going to go ahead and write it that way. Hopefully, other people do, too [chuckles].

And I really think if you like something, go ahead and do it because somebody else is going to like it. With the internet, sure, if you're limited to a small audience, yeah, maybe you're not going to appeal to everyone. But with the internet, if you like it, somebody else likes it, go ahead and do it. Your audience will find you eventually.

He writes that side projects and hobbies are important. Now, I'm the sort of person that I'm going to do that stuff anyway. You couldn't stop me. You'd have to chain me down. But, yeah, I think that's really important. That's the answer to how do you exist outside of a company? Well, you exist outside of a company. Do other things or find the things that are interesting that you do

during your day job, and find a way to extract those, whether it's write about them or make a little toy that somehow demonstrates that.

I'm going to skip to eight, which is "Be nice." That's huge. Man, I am so thankful that social media did not exist when I was younger.

JARED: [laughs]

DAVE: Because I was such an idiot, especially socially. Do you feel that way as well?

JARED: Yeah. I think I'm a bit younger than you. So, as I was coming into, say, my late teens, early 20s, social media was starting to take hold. And I'm glad that I deleted all of those accounts, and nobody can find them now.

DAVE: [laughs]

JARED: Because I had different opinions than I have now. And we were all just figuring out how to navigate that world. And it's a whole different set of problems for young folks now.

DAVE: 100%. And I still can't quite help myself. I still do an occasional hot take, if you will.

JARED: [laughs]

DAVE: But I will say this. I always try to punch up, if that makes sense. I can punch at Microsoft all day long, and they're going to be okay. A little indie game dev or, you know, somebody that just wrote an article, yeah, that could get back to them, and that could really ruin somebody's day. So, I'm so thankful that I learned those lessons before [laughs] I had any sort of what you would call an audience at all. And, you know, I have a very small audience, and I like that.

But then the last one that I think is most relevant here is Chapter 6 that Kleon's got, which is, he calls it "The secret: Do good work and share it with people," and that's literally...that's Ziglings. You know, it was...well, jury's out on whether it's good work, right?

JARED: [laughs]

DAVE: But I'd like to think so. But no, doing this stuff, share it. It was a little scary when I put this out because I had no idea what I was doing, but it was well-received. And even if stuff isn't, a lot of things just crickets. Some of the things I put the most work into I don't get feedback for months, and months, and months, and that's okay. But if you keep doing it, you know, people will come.

The other thing that I'd like to mention is there's this quote that I think about a lot in terms of writing stuff, whether it's something like Ziglings that's more interactive, or just an article. And it's Steve Yegge in 2005 has this blog post, and this is in the midst of a whole flurry of blog posts

that he is writing. And his quote is: "Often I'll get discouraged because I feel like I'm writing about things that have already been discussed into the ground by others. The thing I have to remember is there's a right time to learn something, and it's different for everyone."

And I think along with that, and maybe I'm getting this also from his blog post, which by the way is titled “You Should Write Blogs [laughs].” In addition to that, and I've seen many other people express essentially the exact same concept, is that it's not just that there's a right time, but also there's a right voice for everything. So, even if you think that this has been just drummed into the ground, it's common knowledge; everybody knows this, if you write something, nobody has ever written it like you have before. And your brain works more like somebody else's out there than all the other brains. And so, to your amazement, if you think you're not good at this, to your amazement, you will help someone learn something that they thought was unreachable because you are finally the one that broke through.

The other thing I've noticed is, if I read something multiple times from multiple different authors, the explanations get mysteriously better as I go. And I think it doesn't matter which order I'm reading them in. I think just reading it multiple times helps from different voices. So, again, adding more voices to the mix means that there's something out there for everyone.

And also, of course, the factor where you write, you learn. So, you really realize whether or not you know something as you write. And that process, that thinking process of presenting things for others really helps straighten out your thinking about stuff. That's such a valuable [laughs] process, and you get so much better at it as you go.

In fact, not only is repeating stuff that you think other people have written okay, repeating yourself is something I'm now learning is okay. This is kind of a fairly new revelation for me [laughs]. When I look back at some of my favorite authors, I realize now that they repeated themselves a lot. Some of my favorite authors would reuse...actually, if you look at the children's author Roald Dahl, for example, famous for “Charlie and the Chocolate Factory,” man, that guy reused stuff constantly. Or how about Richard Scarry with his characters? He didn't make up a whole new Busytown every time he trotted out a new book, so people don't mind if you revisit your greatest hits.

And one of the things I'm most pleased with that I've written was a summary page. So, after I did that, Assembly Nights 1, I did...I call it Season 2 of Assembly [laughs] Nights, and I wrote another fourth leg. And it's a really silly, weird fourth leg, and it was truly just an experiment. And I did these ongoing devlogs, and I wrote about it in several different ways.

So, by the time I was done with it, I thought, "Well, gosh, how do I even write about this? I've already written everything. Anything I write will be just repeating myself." I went ahead and did it, and as I was doing it, I thought, "This is the best I've ever described this [laughs]." And by the time I was done, I thought, "Wow, this conclusion page is actually what should be the introduction to this thing. This is by far the best I've ever explained this." And it's fun. I managed

to make it fun because I knew what I was talking about, and I could make these cute, little examples and things like that because I was so confident about what I was talking about, finally.

And so yeah, repeat yourself. Write about things multiple times. And I've seen bloggers that do this as well. They revisit topics, and, again, you're hitting it from a different angle, and you're getting better at it every time. So, that repetition is really okay. And, in fact, it's arguably part of that process of getting better.

JARED: I think let's leave it there. Where can people go to read your blogging and follow you online?

DAVE: Ratfactor.com, and from there, you'll find links to Mastodon, where I mostly post weird, little sketches, watercolor drawings, and I boost cute pictures of birds. And you'll also see my RSS feed, which is really how you'll consume what I do as a blog. Otherwise, it's just a big, sprawling website. You'll find the cards section, which is this enormous, sprawling wiki, and I invite anyone to go sort of get lost there. But, yes, to follow RSS is definitely what I recommend.

JARED: Awesome. I love RSS, so that's perfect for me. Thank you so much for coming on the podcast.

DAVE: Thank you for having me. This has been really cool.

JARED: Definitely going to have to have Dave on to do just, like, a strictly Forth episode at some point because I really want to explore that language and what it has to teach me about programming because it looks tremendously interesting, and his writing on it is really, really cool.

As usual, go follow us on Mastodon or Blue Sky, or LinkedIn. You'll find links below in the show notes.

This episode has been produced and edited by Mandy Moore.

Now go delete some...