EPISODE 974

[INTRODUCTION]

[00:00:00] JM: The software category known as no-code describes a set of tools that can be used to build software without writing large amounts of code in a programming language. No-code tools use visual interfaces such as spreadsheets and web-based drag-and-drop systems.

In previous shows we've covered some of the prominent no-code products, such Airtable, Webflow and Bubble. It's clear that no-code tools can be used to build core software infrastructure in a manner that is more abstract than the typical software engineering model of writing code.

No-code tools do not solve everything. You can't use a no-code tool to build a high-performance distributed database, or a real-time multiplayer video game. But they are certainly useful for building internal tools and basic CRUD applications. We know that no-code tools can create value. But how do they fit into the overall workflow of a software company? How should teams be arranged now that knowledge workers can build certain kinds of software without writing code, and how should no-code systems interface with the monoliths, microservices and APIs that we've been building for years?

Shawn Wang is an engineer with Netlify, which is a cloud provider that's focused on delivering high quality development and deployment experience. Netlify is not a no-code platform, but Shawn has explored and written about the potential of no-code systems. Since Shawn comes from a code-heavy background, he's well-positioned to give a realistic and reasonably unbiased perspective on how no- code systems might evolve to play a role in the typical software development lifecycle.

Also, Shawn is just a fascinating guy and an interesting speaker. He's only been programming for a couple of years and he's learned very rapidly. So there's also a lot in this show about how to educate yourself on software subjects.

We're looking for a writer. If you are interested in writing about computer science and software engineering and earning a little bit of extra money, we have this part-time writer role, and we're also looking for an operations lead. If you can help us make our business run more effectively, then we'd love to hire you for a part-time operations lead.

If you're interested in either of these roles, send me an email, jeff@softwareengineeringdaily.com – And I hope you enjoy this episode.

[SPONSOR MESSAGE]

[00:02:38] JM: This podcast is brought to you by PagerDuty. You've probably heard of PagerDuty. Teams trust PagerDuty to help them deliver high-quality digital experiences to their customers. With PagerDuty, teams spend less time reacting to incidents and more time building software. Over 12,000 businesses rely on PagerDuty to identify issues and opportunities in real-time and bring together the right people to fix problems faster and prevent those problems from happening again.

PagerDuty helps your company's digital operations are run more smoothly. PagerDuty helps you intelligently pinpoint issues like outages as well as capitalize on opportunities empowering teams to take the right real-time action. To see how companies like GE, Vodafone, Box and American Eagle rely on PagerDuty to continuously improve their digital operations, visit pagerduty.com.

I'm really happy to have Pager Duty as a sponsor. I first heard about them on a podcast probably more than five years ago. So it's quite satisfying to have them on Software Engineering Daily as a sponsor. I've been hearing about their product for many years, and I hope you check it out pagerduty.com.

[INTERVIEW]

[00:04:04] JM: Shawn Wang, welcome to Software Engineering Daily.

[00:04:07] SW: Hi. Thanks for having me.

[00:04:09] JM: Pleasure to have you. This has been a long-time coming. You happen to be in town for the No Code Conference, and we're going to talk about a lot of things. But I think that's a good jumping-off point. why is no-code interesting to software engineers?

[00:04:24] SW: I think that's interesting on some kind of two – Of two sort of broad dimensions. So the first dimension is that I think there is a lot of undifferentiated heavy lifting that we do as software engineers, where we kind of code the same solutions for ourselves over and over and over again. Obviously, we tend to have like a not invented here syndrome, where the stuff that we do, we try to make sufficient. But really we should be pawning that off to someone else.

So no-code in that perspective is kind of using someone else's platform library. So as engineers – And particularly I'm from the JavaScript ecosystem. I'm very used to employing other people's libraries and code for that. Why stop there, right? Why stop at APIs thing? There's this whole rise of the API economy. Why stop there? Why not have them create the UIs for me to manipulate that code? To me, that's a much higher plane of productivity than me directly manipulating strings in text. So that's the first dimension.

Then the second dimension is I think it's an opportunity for developers to serve other people. As developers, a lot of the way that we view solutions and problems out there in the world is if there is more problem, if there's a problem up there, then solution is more code. We tend to heap a code on top of solutions. Unfortunately, that's not very accessible to a vast majority of people out there who don't code.

If we just kind of serve the no-code market, we tend to make it more accessible to the vast number of users. They're actually less demanding, I feel like. So they just have a certain few use cases that they really want to serve. I think that's really interesting. My day job, I work at Netlify, and we are a developer-facing company.

But there are a ton of use cases and a ton of complexity to manage. I do look at the no-code space as like a different target audience with much more standardized simpler demands and they're more willingly pay, because developers are famously not willing to pay for stuff. As I think

from a business opportunity, no-code is very interesting. For developer productivity perspective, no-code is super interesting

[00:06:47] JM: I think the thing that – If I think back to the conversations I've had with people who are concerned about no-code or who are not – Maybe they're curious about it, but they haven't actually tried it out or they're afraid of trying it out. My sense is that they're afraid that when they get to the point where the no-code solution is too abstract, too coarse-grained, they're afraid they won't be able to duck down into the code.

React, for example, there're lots of open source React components. If a React component isn't performant enough for you, you can just fork it and like rewrite it, right? With no-code, that may or may not be the case, right? Because this is like close source proprietary code and people are afraid of that.

To what extent is that a valid concern?

[00:07:38] SW: I think it is very valid, and you should be worried about platform lock-in through that. I think poorly designed no-code platforms will happen and people will get burned, but that doesn't mean that it's impossible for a good abstraction to form where it's very clear what the responsibility of the platform is. If you need to eject, there are ways to do that.

So Webflow, for example, it does export to HTML and CSS and it actually is a maintainable code that you could take over yourself. To me, that's very attractive. I think we're still learning how to do these right, and the more we do it and the more we understand where the seams lie, we can start to understand where the convenience stops and the complexity starts to takeover that we should in-house that.

But I think a lot of the times, most of the times we're just like raising that concern preemptively to stop ourselves from even trying it out in the first place, and I think that's wrong. I think to have that sort of vision as a developer that, "Oh! I'm better than this. I'm better than a no-code tool, because I can code everything myself." What about this, this, this, this edge case? Which you maybe never hit, because a lot of times it's like what if you're just doing an MVP? You should

just be delegating to a simpler commoditized intuitive visual platform to build out your functionality.

If you do need to eject, you have that skill, and I think that's something that maybe software developers underappreciated. We are trained to think in abstractions. We're trained to take different tools, assess, analyze and put them together to produce business value. I don't view no-code as any different from that category of tools. They're just targeted at a broader audience than just you.

[00:09:17] JM: If we look something like Twilio, Twilio fulfills a very specific purpose. It does your telecom stuff. It sends your text messages, for example.

[00:09:27] SW: Or email.

[00:09:29] JM: And email and a lot of other things. But if we just look at like the text message use case. Okay. Twilio is fulfilling my SMS notifications or two-factor authentication through SMS. It's very well-defined, and I can build my application with the knowledge that that is a dependency that I am paying for, and it's proprietary, and I can be comfortable with that, and I am comfortably abstracting away that that duty to Twilio.

No-code, it's fulfilling a broader set of characteristics, and I think people are afraid of there're being too many things that it's fulfilling. Do we have a concise perspective for the duties that you want a particular low-code platform to satisfy? Is it the UI layer? Is it like your Shopify store? What are the barriers or what are the borders around which the no-code solutions are satisfying?

[00:10:32] SW: So I don't think there is a definitive drawing of the lines yet. I think a lot of these people are just trying to figure it out. By the way, I'm not an expert in this. I'm very much dipping my toe in the water as well. My perspective is that basically everything can have visual layer to it. I recently wrote a blog post on my blog about this where it's a fact.

[00:10:55] JM: No-code is a lie.

[00:10:57] SW: No-code is a lie. It's a fact that the API economy won. Stripe, Twilio, everything as a service that has won, but why stop at just the API layer? The API layer still requires people to bring up extra codes. Still requires documentation for people to make use of it, right? So why stop there? Why not add an additional graphical user interface layer?

I kind of call this the GUI economy that lets you hook stuff up within –

[00:11:24] JM: GUI, G-U-I, graphical user interface. The GUI economy.

[00:11:28] SW: Which to a lot of people is the ultimate vision of no-code or low-code, right? You're formatting the business logic or you're creating all these commands are scheduling actions or whatever through a visual interface that's purpose built for that. So you cannot make mistakes. It's easier to discover whatever capability is out there, and you're just spending this time in your terminal and more time, and you're just leveraging things that have been proven and tested to work.

So I am definitely on the more permissive side of this debate where I think that everything can be improved with an extra no-code layer, which like we have this arbitrary thing of like what programming looks like, which has moved over time and will continue to move beyond where we are in this point in history.

So one really key example I can sort of provide to you, again as a developer, is that the Vue ecosystem is one of the more sort of beginner-friendly ecosystems out there in the JavaScript world, and they used to have a CLI, a command line interface, where you're typing commands in your terminal, and that kind of looks a little bit – It's a little bit inaccessible and it's hard to discover what's available out there.

So they're investing in a graphical user interface for this CLI. It's called the Vue CLI UI, and it's like, "Why would you have UI for your CLI?" It's purely because it's easier to learn and access and discover and it's more intuitive –

[00:12:58] JM: Wait. Sorry. There's a user interface for the Vue CLI?

[00:13:03] SW: Yes.

[00:13:05] JM: What does that have? Like drop down menus and auto complete?

[00:13:08] SW: Everything, and visualizations for your builds and all that.

[00:13:11] JM: What is it? An Atom or a –

[00:13:13] SW: Electron.

[00:13:13] JM: Electron app. That's pretty cool.

[00:13:15] SW: Yeah. Once you start looking at this, you see it everywhere. So part of my article that I wrote about this, like – Okay. So no frontend developer codes animations and Bezier curves without looking at the visualization of the animation. There's another movement where we're trying to visualize our state machines in frontend development, and obviously you need a visualization for that. So there are all these visual assistant tools –

[00:13:38] JM: Why do I need a state machine for UI development?

[00:13:41] SW: Can you repeat your question?

[00:13:41] JM: Why do I need a state machine for UI development?

[00:13:44] SW: Because there are a lot of states, and you want to control the transitions between them. Either you are controlling them implicitly and doing a lot of if-then-else, or you have a formal explicit state machine where you can actually visualize it and check that. You're programmatically – That you're definitely thinking about all the edge cases that are going through them.

[00:14:06] JM: Interesting.

[00:14:08] SW: There's a lot of states in UIs. I don't know if people –

[00:14:09] JM: It's true. It's like are you on mobile? Are you on desktop? What the size of the window? Where are you in the window? Which parts of the form have you filled out? That's a ton of states.

[00:14:20] SW: Is text selected? Are you – Scroll to this certain height? Yeah.

[00:14:26] JM: Have long have you been on the page?

[00:14:27] SW: It's something asynchronous in flight. Yeah. There's just a ton of stuff. That's a huge movement in React.

[00:14:33] JM: And people literally use like finite state machine logic to –

[00:14:36] SW: Yes.

[00:14:37] JM: Wow!

[00:14:38] SW: Yes, and that's one of many examples. Why is VScode been so successful over VIM and other editors? We want a visual interface. Even when we're coding, we want to click around. So it's a very fundamental human nature, and I think when we develop software, we tend to have a fixed idea. Basically, whenever we start, it's when we freeze our conception of what software is.

[00:15:03] JM: Right. That's so true. That's so true. Actually, that calcifies something I was trying – I was thinking about like – When I was thinking about this interview, because you start programming – Well, like 3 years ago? 2 –

[00:15:14] SW: Two years ago.

[00:15:14] JM: Two years ago. That's incredible, man.

[00:15:16] SW: Thanks.

[00:15:17] JM: And you have a pretty fresh perspective. You've really gone deep. You've been just engorging yourself in information, which is really admirable. That's my style too. I just love people who just consume tons and tons and tons of information at a fast rate, and you've done that.

Because you are not normal – You've reached a level of maturity in your career very quickly and yet you are not normalized to where – I began my career when like Java was the thing. Java was the only game in town when it came to enterprise applications. I mean, there's a little bit of Python, a little bit Rails, but it was mostly just Java.

The way that frontend people program was like sublime text, and you're not using sublime test. You're using like text Wrangler, or like something, some text editor, but there is certainly no IDE you would use. Maybe WebStorm if you are like a weirdo. But now, like you said, you're using an IDE, or you're insane, right? That's the case, right? I mean, or are there still like text editors?

[00:16:21] SW: We're generalizing a huge [inaudible 00:16:23] of operation.

[00:16:24] JM: That's great.

[00:16:25] SW: But, yes. In general, I do come at it with – So my philosophy professor calls this unencumbered as I am by any knowledge of the matter. Then he will like put up his criticism, and it's correct. It should make sense from scratch, because that is kind of reasoning from first principles rather than reasoning by comparison and similarity and what you had, like your reference point from before.

So I don't know what the goal of that is. I mean, I do think that as I've seen this like sort of – And I've understood and absorbed what this no-code movement is about. I think that that's a very big motivation for why people are seeing so much potential for this.

It's funny, because if you look at – So the definitive inspiration for Vlad Magdalin when he started WebFlow, which you had him on the podcast before, was Bret Victor when you talked about the future programming at the Dropbox Conference in 2011. He was talking about the

state of human computer interaction research in the 1970s. They had all of this mapped out. All of it. None of this stuff that we're doing is new to the people in the 1970s. We just didn't execute on it. It's pretty absurd, because it's pretty obvious like what we humans find more intuitive than not. It's just takes a lot of work.

[00:17:48] JM: What Vlad said in that episode, people have been trying for a while. People have been trying the no-code stuff. It just the tech wasn't there yet.

[00:17:56] SW: Platforms need to improve a lot, and JavaScript has made it a lot more feasible to do things in the client site that previously was just impossible. It's not necessarily the case that we needed JavaScript. We had a lot of this in Flash and Visual Basic and what have you in previous platforms. But I think it's still a worthwhile goal and it doesn't invalidate the overall thesis that a no-code tool is more accessible, and therefore more valuable actually even though it may be less powerful than a full-code approach.

[00:18:30] JM: I think the term you coined, the GUI economy thing, I think that's a legit way of looking at it. That idea that you've got this wide buffet of different APIs to select from, and the way that people have been thinking about it is like, "Oh yeah! You get tons of APIs. You write some JavaScript glue code, and then you write an entire user interface, and all you're doing is like cutting and pasting React components. It's that easy." That's not easy. That takes a lot of time, and certainly it's better than writing like a bunch of custom backbone components or even custom React components before there was a bunch of ones that you could take off the shelf.

But what you're pointing out is that why even do that? We have pretty well-defined primitives for building UIs. Why go as low-level as piecing them together in terms of React components? Why not do bumper bowling, like use WebFlow, or Bubble, or Airtable, Zapier, like Typeform, whatever the hell like low-code thing you want to use, because like it's still going to take a ton of work. You're still going to have to build a user interface. It's still going to take you a ton of time. You're still going to have to wire together all these APIs. So there's still going to be some defensible advantage to your company that you're building.

The GUI economy, the idea of selecting between these different – So what are the GUIs you're selecting from? You're selecting from like WebFlow templates? What is on market in the GUI economy?

[00:20:13] SW: Oh gosh! This is such a huge debate right now. WebFlow does a ton of stuff. It's not just templates. It's also they have a GUI for CSS. Every single piece of – Every single field in CSS is modifiable in a visual manner. You can like slide and drag and drop and click buttons to center things in CSS, which is a known hard problem in CSS.

But they also have an e-commerce solution. They also have a CMS solution. Those all GUIs on top of an existing API that they provide for people to hook together. Why am I looking up documentation when I could just go click that? That's obviously way easier way faster.

But I do want also make the point – So like the way you're thinking about GUIs right now, you're stuck in this mode of like the end consumer consumes the GUIs, but also the developer conceives GUIs as well. What is all this cloud and serverless and platform as a service movement but no-code for backend, right?

When I deal with Firebase, I'm clicking around and I'm just like configuring my database and I'm setting stuff up. As a frontend developer, I'm so enabled by that because I can just copy and paste and check it in there and then that's it. What it's like serverless?

There's a lot of other infrastructure as code movements where you're sort of declaratively declaring your source. You're not setting up any of that. You're just sort of managing – You're managing your declarative template of the resources that you're using, and everything else is just figured out for you by some compiler approach, some of that source.

So I want to be a little bit more expensive than these kind of end user no-code tools, which is that's a whole promising field in itself and user computing. But also as developers, we use no-code tools in our daily work, and I really want to drive that point across where like as – Again, as someone that works at Netlify, I do use other services as well and throw a bunch of Firebase and AWS. They all have user interfaces that we all use all the time.

[00:22:09] JM: You use Firebase in Netlify?

[00:22:10] SW: No. In terms of personal projects. But, I mean we have a ton of people who use Netlify Firebase for databases and authentication and all that stuff. But like I'm not ever – I was just discussing with one of my friends who is a prominent GS developer. The de facto solution for coding authentication in Node.js is passport.js, and that is regarded as easy. It was revolutionary when it came out. Nobody wants to use it anymore, because we are all sold on this idea of we should farm that out Auth0, Octa, Netlify, Firebase and they will just set up my Twitter off and my email off in a couple of clicks, incognito.

[00:22:53] JM: There's no open source ideologues that are like passport.js all the way?

[00:22:56] SW: I'm sure if you look hard enough you'll find them.

[00:22:59] JM: But that hard and fast, everything needs to be open sourced. That's kind of going away? Because I think –

[00:23:05] SW: It's commoditized.

[00:23:07] JM: It's commoditized. So it's cheap. People don't have a problem with it being close sourced because it's cheap and it's not like lock any. It's like pay us enough – Like, "Hey, we're Auth0. Could you please pay us enough to continue operating your authentication? Just pay us enough. We're not going to screw you over," and that's the deal, and people are comfortable with that deal.

[00:23:28] SW: Absolutely. I mean, the economics and the lock-in of platforms aside, I think that's a perfectly rational thing for developers to be doing, because at the end of the day, the way that you sign in is not going to differentiate your products. So you should find that out to other people. But the point I'm trying to make is that the reason developers are just like flocking in droves to where it's all these dedicated providers is because they are also no-code tools already.

For me, when I use Netlify, when I set up Netlify identity, I'm literally clicking a couple of buttons and I set up Google, Twitter, Facebook off, and that's the way I want to code. I want to spend my time on things with business value, things that matter. Everything else should be no-code for me. I think that is generally true for everybody. Then we can talk about end-user computing and what that means in terms no-code. But it's pretty – Like once you see it, you can't un-see it. Everything is no code.

[SPONSOR MESSAGE]

[00:24:30] JM: Looking for a job is painful, and if you are in software and you have the skillset needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that's a good fit for you.

Vettery is an online hiring marketplace to connect highly-qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies access is exclusive and you can apply to find a job through Vetter by going to vetter.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you.

No more of those recruiters sending you blind messages that say they are looking for a Java rockstar with 35 years of experience who's willing to relocate to Antarctica. We all know that there is a better way to find a job. So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery.

Vettery is changing the way people get hired and the way that people hire. So check outvettery.com/sedaily and get a $300 at bonus if you accept a job through Vettery. That's V-E-T-T-E-R-Y.com/sedaily.

Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:26:20] JM: To help us talk through this emergent space a little bit more, let's talk through another emergent space, one that you're more intimately familiar with, which is the JAMstack. The JAMstack Conference was a while ago. You're just deeply into the JAMstack, because you work at Netlify. You're as deep into it as anybody. How does the low-code ecosystem compared to the JAMstack ecosystem?

[00:26:42] SW: Wow! I think no-code is probably – As a trend as a term –

[00:26:47] JM: Actually, for people who have no idea what it is, can you just give a brief overview of the JAMstack ecosystem, and then we can compare it.

[00:26:52] SW: Sure. So JAMstack is this sort of umbrella term. So JAMstack stands for JavaScript APIs and markup, and that is a movement towards more static assets first as a platform for delivering web apps in general. We think that is faster, more secure and cheaper, more scalable to host apps in sites this way. Netlify is founded entirely on this thesis and will live and die by its ability to serve the JAMstack developers. So JAMstack is a movement that's bigger than Netlify, but Netlify is trying to sort of be at the forefront.

I think most people – By far, most people will be familiar with the GitHub pages, and GitHub pages is where you host your docs. If you're on GitHub and you're trying to host docs in the Python ecosystem, in Ruby, everyone hosts on GitHub pages, and it is the simplest approach, right? You build the site. You built to static files and you host it on a static host provider and it's super cheap and you never really go down, because there's no server to overload.

So that does the whole thesis. I can go deeper into that. But it started five years ago, and I think it's really picked up steam since then. To go back to your original question of how do you compare that to the no-code. I'd say no-code is a little bit newer in terms of like as a meme, as a trend in tech. I think they're very comparable, because JAMstack is no-code backend. A lot of JAMstack players are – Like ZEIT, Netlify, Amplify from AWS, and there're are other bunch of other small players.

But there are just objectively more frontend developers, more people who do JS. More people who wanted to work in products rather than platforms. So they much rather just employ a platform provider like Netlify to do all that while they work on their product. So that's good. That serves developers who are frontend developers and they're more than people who are legitimately full stack. There a lot of people who pretend to be full stack, but they're not.

So then you compare that to the mass of people who are not developers. There are more of those. So I think there is a lot more potential in terms of like raw numbers for the no-code movement. I think building the right abstraction, like we talked about earlier, is going to be way harder, because these people are people who can't help themselves. Once they run into trouble, they're stuck. Whereas developers can figure it out and then they'll send you their bug reports and they'll help you PR the fixes, which is fantastic for us. No-code developers need a little bit more handholding.

I think I kind of draw the analogy a little bit to the design space. So a lot of people and a lot of designers love thinking about this as like the hot new thing among Sketch and Photoshop and whatever else you have. Figma was started in the same year as Canva, which is a toy, a tool, a joke to designers. People laugh at kind of it, because you'd drop in an imagine and you create a Facebook banner and it's good. It's great.

Figma started the same year as Canva. Canva is now six times the size of Figma, at least. It's absurd. Obviously, both companies are fantastic. They're both unicorns. They're both doing great. But there is this truth in there is money in the masses, and it does I think have this like a very interesting set of fractal attention to it where –

[00:30:05] JM: So Canva is easier to use?

[00:30:07] JM: Canva is way easier to use, yeah, if you've tried it out. Definitely use it for your social media. Absolutely.

[00:30:13] JM: Okay. I tried Figma the other day. Figmwa was awesome.

[00:30:16] SW: Right. Figma is a designer.

[00:30:18] JM: It's Photoshop.

[00:30:19] SW: Yeah, but like I think –

[00:30:20] JM: Easier than Photoshop.

[00:30:21] SW: I think tools that let non-Xs do X are very interesting category of tools. So, arguably, no-code tools let non-coders do their coding stuff, and commoditize coding stuff, and that liberates a whole bunch of people.

Netlify, JAMstack, let's non-backend – People who rather not spend your time on backend hosting, undifferentiated hosting stuff, to just let providers do that and then they can then focus on their frontend. I think that's an interesting like 30,000-foot view of like comparing the two different ecosystems.

I'm probably writing rough shadow over a lot of different nuances. But does that make sense to you?

[00:31:05] JM: It does. What code are you writing in the JAMstack?

[00:31:11] SW: So JAMstack is very much you're writing your frontend and then you're writing any API glue code that is needed to piece your different APIs together. Does that –

[00:31:25] JM: Yeah. Sure. That's it.

[00:31:27] SW: Yeah.

[00:31:29] JM: So is there a typical JAMstack or the whole idea is just that there are lots of APIs and the markup is – I guess it sounds like basically the difference between the JAMstack and the no-code stack, is in the JAMstack, you have to actually write glue code. You are actually writing some backend glue code. Whereas in the no-code stack, you have to find providers that can either give you something that looks like glue code, like Zapier, which is kind of a glue code

as a service, or you have to have a GUI, one of these things from the GUI economy that kind of comes with the custom backend glue code that you need preconfigured. Then the other difference is in the JAMstack, you are always making markup. So you are always designing your own GUI. But it kind of seems like in the limit, these are –

[00:32:31] SW: Similar.

[00:32:33] JM: Heavily overlapping. Heavily overlapping paradigms.

[00:32:37] SW: Yes, absolutely. I think a lot of JAMStack users – Again, I'm trying to make that analogy from the foundation of these are just two different degrees of no-code, right? One you kind of stop at the backend, maybe the other you go all the way, and it's not clear which is the better approach, but they're all definitely better than doing everything yourself.

[00:32:55] JM: It kind of seems like you can go from no-code to JAMstack, but like JAMstack to no-code, I don't think I have seen an example of that. I've seen like no-code JAMstack fairly often, like the Moonlight. I don't know if you listen to the Moonlight interview, but like that was really interesting one. This company Moonlight Work, where they start – I mean, it's like led by a great developer. I think Emma is also a developer. I think two developers. At least one very strong developer, and they just pieced – It was one of these like piece together a type form and a bunch of different backend things, because that's the fastest way of validating it. Then once they found traction, build a bunch of custom infrastructure and gradually replace the no-code stuff.

But I don't see anybody, again, doing the opposite, where you start with mostly backend situation and then you figure out what your frontend is going to look like with markup and then put WebFlow in front of it. I've never seen that.

[00:33:54] SW: Yeah. I mean, it's probably a thing about supersets, like JAMstack is probably a superset of this overall – The abilities that you have in the no-code space, but that's just what code is. Code is so malleable and flexible that it can do literally everything. Sometimes that's good. Sometimes that's bad. A lot of times it's bad. I don't necessarily see a bias in terms of like

drawing that. You can transfer from one side to the other. Yeah, I don't think that's necessarily better or worse. It's just different.

[00:34:25] JM: Are enterprises building JAMstack applications?

[00:34:29] SW: So they are, and I think they're in different stages of adaption. So we kind of split things in terms of sites and apps, and sites is much clearer because you do have static content and you're just serving it, and that should be as cheap and secure as possible with minimal downtime, yadi-yadi-yada. Obviously –

[00:34:49] JM: Enterprises have plenty of static content documentation.

[00:34:50] SW: CDN. Yeah. So one of our sort of primary examples of this kind of thing is Citrix, which move everything on citrix.com. TriNet, which moved everything on trinet.com on to the JAMStack.

Oh! I remember –

[00:35:04] JM: Is it china.com?

[00:35:05] SW: TriNet.

[00:35:06] JM: TriNet. Okay.

[00:35:07] SW: Yeah. But everyone is familiar with TriNet. It's a hustle name payments and HR stuff. So I remember the third category now, e-commerce. So there're sites and then there's e-commerce and then there's apps. E-commerce is like this weird in-between thing where like it's kind of a site and it has to be – Performance is hugely important because of people being very fickle and bouncing if they don't load in some certain amount of time. SEO searchable is very important. But then there is a lot of dynamic activity that has to happen. You have to have shopping carts. You have to have people logging in and remembering purchases and so on and so forth.

So e-commerce players at the JAMstack Conf, we had Loblaw, which is one of the bigger retailers in Canada moving their entire site over and talking about their –

[00:35:53] JM: From what?

[00:35:54] SW: Performance benefits. I think it was just like a self-hosted, like sort of EC2 type solution.

[00:36:00] JM: Oh wow! So it moved from like some EC2 node app to a Netlify app?

[00:36:07] SW: I don't think it was node. I think it was some other language. I didn't really ask too many questions because –

[00:36:14] JM: They re-platformed.

[00:36:14] SW: They re-platformed to the JAMstack. For that, the performance is the key consideration, because for e-commerce every second is like $1 million or whatever. Then the final case is the app use case, and that is like can you deliver a progressively enhanced static app that rehydrates into a dynamic JavaScript app on the frontend? That is covered by PayPal, which also recently talked about how they moved to the JAMstack at QCon.

[00:36:42] JM: The e-commerce example. So in that example, if they're going from a dynamically built – Like I land on the website and the website is being dynamically built and it's slower because of that. They're moving from that to a website that is basically precompiled cached, and therefore is going to deploy or it's going to serve user requests much faster. So it's just latency. So the build process gets frontloaded and pushed out to the CDN, rather than I land on the website, I add something to the shopping cart, I search for something else and all these things are like a very heavy request response that requires building an entire webpage and sending it back to you. Instead, you just have this situation where the website is built and, yeah, pushed out to the CDN.

[00:37:37] SW: Yeah. I like that matching of – A lot of this stuff, a lot of like page visits are people who are never going to pay you, right? Usually, you should match that with a more static

approach, where it should just be served free from the cheapest infrastructure that you have. But as the have more activity, then you match it towards something that scales up.

So JAMstack is kind of like the counterpart of serverless, where we really liked to scale things up. You don't have to use serverless. You can use microservices as well. But it definitely – I like that pairing of like for users, use commoditized static assets as you engage, as you as you start interacting with our services more.

[00:38:15] JM: What application of serverless are you talking about here?

[00:38:18] SW: For example, we're talking about the e-commerce app? It's unacceptable to me, that let's say it's Black Friday and a million people land on my page at once. The people who are not paying for anything, and they're just checking out your inventory, could bring down my server. I'm talking like pre-JAMstack. If you're not on a JAMstack. If you're serving everything on requests and you're just like looking all these concurrent users. You don't know which one is – Someone who's going to convert. You're just serving this content on request.

So like they could bring down your server just because you have your maximum concurrency of limits and all that. So, to me, that's very interesting to say that, "Okay. I'm going to just pre-render all these static content, and that will always be up. Anything that needs to scale, I'm just going to whittle that down to the smallest possible atom of like just that API response." Instead of like regenerating the entire page all at once, which is what you would do in sort of the pre-JAMstack paradigm. Does that make sense?

[00:39:14] JM: It does. Now that we've gone down this track, I want talk a little bit about the pre-compilation step and kind of the JAMstack variety of ways of getting an application delivered to the user, like created and delivered to the user. So I don't know if you listen to the – There's a Gatsby episode we did.

[00:39:36] SW: I didn't.

[00:39:37] JM: Okay. You listen to that one. We did an episode with Gatsby a couple of days ago. I don't really understand how Gatsby works. I don't understand why so many people use it.

Can you help me understand Gatsby and tell me like how widespread is the usage and how are people using it?

[00:39:55] SW: Okay. I think there are probably definitive numbers out there on how widespread Gatsby is. A lot of these people actually track like percentage of top million users. How many of them are traceable as like Hugo, Gatsby, Jekyll, all these site generator? Using I think builtwith.com as the authoritative trackers on those things. I don't have them at-hand. So I can't really comment.

I think in terms of React, which – I kind of place it as this like JS is huge. It is the biggest programming language apart from Excel. We won't talk about Excel. But JS is huge. React is huge within JS. Gatsby is huge within React. So it's kind of like top-top-top, but it is smaller concentric circles.

So it's big enough that like the React docs use it. A ton of like – You just go look on their showcase, gatsbyjs.org/showcase.

[00:40:46] JM: You would describe as a build tool?

[00:40:49] SW: Yes. It has a build tool in it. So does that make sense?

[00:40:53] JM: It does.

[00:40:55] SW: A lot of JS has different jobs to be done in it. Gatsby has multiple jobs to be done, and I think you sort of talked about that in your episode. One of them is providing that default webpack configuration, and then allowing people to modify it, but providing that good default, where you can build performant React app, which is a pain point for people. That's why –

[00:41:18] JM: What's a webpack configuration?

[00:41:20] SW: So webpack is the predominant build tool in JS. There are others, like Roll Up in Parcel, which are gaining steam. But webpack is by far the biggest, and it's hard to configure. A

lot of people don't want to sort of set it up manually just because there're a lot of small details to get right, and no surprise, web performance is really hard. Gatsby is the whole pitch of like use us and we will give you the best performance.

[00:41:41] JM: What kinds of configuration options does it give you?

[00:41:45] SW: I myself don't even know everything. But I'll just give you some simple ones, like notification of JavaScripts. Long variable names, helpful during development, but why are you serving them over the wire when it just compiles on the code? Stuff like that. Really basic blocking and tackling stuff, webpack is really good at. Then Gatsby tries to just be that best performance, like best-of-breed performance.

Simple things like when you're including like an image, a lot of times your including the raw image, which is like 60 megabytes and you're serving you in like a 200 pixel square. Obviously, you should down sample it, right? Where are you going to do that? You can sort of do that ahead of time, or you can just stick it in your webpack config. All of these stuff is being stuck into the build tools.

Guillermo Rauch, which you also interviewed, kind of views this – This is hugely contentious amongst some parts of the JS community, because a lot – JS was not meant to do any of this. Brendan Eich was like, "We just want to put some script stuff into the HTML, and that's should be all it does."

A lot of people learn JS from View Source. I would just right-click and View Source on the page. having a build step in between obfuscates a lot of this code and makes it a little bit more inaccessible, which is a problem, but the benefits that have been unlocked from putting all these build tooling for ultimately the vast number of users that will never see your code, but they'll see the user impact of your code, which is load speed and like performance and all that. Guillermo calls it a Pandora's box, like it's open. Deal with it. We have to have build tools now.

[00:43:23] JM: Right. Okay. So this really useful to me. So now I understand that like, basically, the Javascript that you write, generally speaking, it could load on the user's browser. You could ship it to the user as the code that you write.

[00:43:40] SW: Which is guaranteed to be suboptimal.

[00:43:42] JM: Guaranteed to be suboptimal, guaranteed to be slow. Webpack is a tool that gives you configuration options for how to make that code look differently, be packaged differently, be delivered differently. So maybe it's a smaller blob of code. Literally, a small amount of resource that you have to ship to the user's browser. Maybe it's shrinking the size of the images without creating any kind of lossiness and blurriness or compression issues. You can imagine some number of things –

[00:44:18] SW: Someone's probably yelling at me now. So I probably should shout out the bigger benefit, which is Webpack's original claim to fame, which is modules in JavaScript. JavaScript was originally one giant file and you just like – You grab around this and you look for the things. If you wanted to interoperate between different scripts, you probably stick things on the window and just have it be a magic global that you kind of import somewhere. That was the way of things were for 20 years.

The module system came along and there are different proposals, but it eventually [inaudible 00:44:46] modules one out. Webpack was one of the earlier, earlier implementers of those. There were others before, like RequireJS, but those are kind of on their way out now. Webpack brings modules. We want to write maintainable apps on the frontend. You want to take JavaScript from – It used to be a toy language ad you want to write series apps. You probably should have different files with different responsibilities and different names and have some sort of build tool that cap them altogether in a sensible way. So that is chief benefit, because it invented the import-export syntax. Then everything else, you sort of stick into the build lifecycle to optimize your build. But having modules is important.

[00:45:23] JM: Right. Okay. Then Gatsby I guess is also useful because it can reach out to a bunch of other resources and pull those resources in prior to the webpack.

[00:45:35] SW: The content mesh approach.

[00:45:37] JM: Content mesh.

[00:45:38] SW: Which is more ambitious than – So I do want to make clear. Netlify is not – You don't have to use Gatsby. You use Netlify, and Netlify has a bunch of there – Is very much tooling agnostic to all these. So there's Jekyll, there's Hugo, there's Nuxt. There are a whole bunch of other tools that are usable with Netlify, and Gatsby is just one of them. But Gatsby is probably the most ambitious in terms of its data pipelining, because a lot of static site generations just pull from markdown and generate static pages, and that's pretty much all they did. Gatsby is able to put any arbitrary data in – As you talked about in that episode. Then sticks it through a GraphQL layer just to be more accessible, and is also a trendy and sexy.

[00:46:22] JM: Right. Moving on, it's late 2019. How do you see the difference between the React ecosystem and the Vue ecosystem?

[00:46:36] SW: Ooh! I should disclose my biases. My career is basically build on React, and I'm a moderator of the r/reactjs subredit. So I do have my cards a lot in one side of things. I did start out in Vue. So I found Vue easier to get started with. The main reason I switched to React is because there's objectively more jobs. If you go on to the Hacker News jobs trends scraping page, React has been on top for 30 months. It's ridiculous. There's this meme of like there's a new frontend framework every few days. It's not true anymore.

[00:47:13] JM: No, it's not.

[00:47:15] SW: Things have come down a little bit. I think what is shaking out is Vue does have a sort of more community feel. Again, I'm probably going to piss off a bunch of Vue people, but they are my friends and my boss, [inaudible 00:47:26], is a very prominent member of the community. So Vue has more community feel, because it's started by an independent Hacker, Evan You, who you probably should have in the pod at some point.

[00:47:36] JM: I already did.

[00:47:37] SW: Oh, you did? Okay. Cool.

[00:47:37] JM: Need to do it again.

[00:47:39] SW: Yeah. Totally. He's doing awesome things, especially with Vue 3.

[00:47:43] JM: He's an inspiration.

[00:47:43] SW: Yeah. So they have a 30-person core team and they all do different pieces, dissent pieces and none of them reports to any major company. Although I think GitLab is one of their biggest users.

[00:47:54] JM: Oh!

[00:47:55] SW: GitLab.

[00:47:55] JM: No. I said ooh!

[00:47:57] SW: Yeah. I mean –

[00:47:59] JM: And China.

[00:48:00] SW: And in China.

[00:48:00] JM: Tons of Chinese companies.

[00:48:01] SW: Exactly. Whereas React is famously from Facebook, and there are people who are just morally against that. You can never change their opinion. But it's hard to argue against – Before anything in React is released, it's tested at-scale by Facebook.com. That is just true. The concurrent mode stuff that you talked about with the Gatsby folks, that has just been released. It was tested in production in FB5, the big rewrite of Facebook.com that just rolled out. That's before they released it to us.

Usually, it's kind of the opposite. You'd kind of have to like release stuff in open source. Then maybe if you're lucky, some big corporation will pick it up for you. This way it's kind of like the

reverse. People who are employed at Facebook, they have a thousand React developers. They're testing out this API for me and I only just get to benefit from what they find out.

So I think that's kind of how it breaks down. Obviously, the money is super important. I think that people who use React at-scale is super important, and that's a huge ecosystem benefit that React enjoys for now. I think that the cross-platform focus of React is like React is not just React. React core team is 8 people. The React native team is 20. Then React web is like same size.

But like React is not about React web. React is about the universal cross-platform language that you can author the same codebase potentially on your mobile apps, Android, iOS and web. Very bold, but it's at least revolutionized frontend in terms of proving out that components are the way to go. So Vue, [inaudible 00:49:40], Angular all move to its components now.

Then immediate mode versus retain mode in terms of the way you want to declaratively author frontend experiences. That's also been pretty much proven out as well. So React has a lot of credit to its name because of all these innovations. I think it's got more to come. It's a little bit – The easy fruit has gone. The low-hanging fruit has gone, and now it's trying to pitch this concurrent mode thing, which is so hard to explain. I do it. I do talks on this, and I find it hard to explain. It just is what it is. UI, if you want to get it right, if you want to get intuitive. A lot of us, as users, we take it for granted. But there are a lot of people sorting the details behind the scenes.

[00:50:24] JM: Yeah. This can be exciting to see. I mean, the UI layer is really getting up and it's really getting technical. Talking to the Flutter people – Flutters is very technical. I don't think that's really – That's not a web layer. That's just a cross-platform.

[00:50:44] SW: The React team draws inspiration from all UI interface experiences. Other interfaces are drawing from UI. So SwiftUI is like kind of like the big new thing in Apple app development, and that they directly credit React for their API.

[00:50:57] JM: Oh, really? That's great.

[00:51:00] SW: Yeah. Pretty huge. Google also has – Apart from Flutter, they have another initiative. I can't remember. Go ask Leland Richardson, who left Airbnb to work on this at Google. I think it's like Jetpack, or I don't remember the name. It's not.

But I want to say one thing. I think we're not done yet in terms of movement. I do want to sort of throw the bone out there in terms of like that whole like what happens in 5 years question. There is a movement where it's more compiled approaches. Flutter is a compiled approach, whereas React is all-in on this JavaScript runtime, and then you have bridges across.

The ongoing research in all frontend frameworks right now, except for React, is more of this compiled approach, two words, whatever target device is being run on. So Angular is exploring Angular IV, an perhaps the Glimmer VM, which they're always talking about. Vue is also exploring the compiled approach, and Svelte is the new framework that I'm super excited about that is built from the ground up to be a compiler, and it compiles from the component authoring format that you're writing in to direct DOM instructions, which is obviously smaller and –

[00:52:07] JM: What's the difference between React having server side rendering and a compiled approach?

[00:52:12] SW: So that's an interesting question. There is a very clear difference. So server side rendering is you're rendering all these HTML out, whereas the compiled approach is saying, "All right, this parts of the runtime, you don't use. So I'm going to take it out the JavaScript bundle itself." One affects the HTML that the user see. One affects the JavaScript that is ultimately generated as the final out of the app. Does that make sense?

[00:52:36] JM: No. I don't think I understand that.

[00:52:37] SW: So ask me the same question.

[00:52:41] JM: We're talking about compiled approaches. So like whatever. The degree to which you can compile something for the web, that's just like getting it down to JavaScript and HTML, right? That is what compilation means on the web.

[00:52:56] SW: Yes.

[00:52:56] JM: So if we're talking about Vue, that's compiled as you can get. If we're talking about mobile, then you can get into like ARM – I think, like ARM instructions, and I think that's what Flutter does. So you can compile your UI down into like machine code.

[00:53:14] SW: I think you have a misperception in what server side rendering does then.

[00:53:17] JM: Oh, okay.

[00:53:17] SW: Because server side rendering is purely generating HTML. A typical single page app in the HTML JS CSS frontend space is a blank page with a single div. Then the JavaScripts that is code in one of these frameworks hydrates that into a full domino with all the other HTML stuff. So this takes –

[00:53:40] JM: On the client.

[00:53:41] SW: This takes extra work and it's slower to show something up on screen. So service side rendering is to say, "Do this on the server." The HTML that you send over already has all the extra dominos that would have been generated. So you're immediately able to – First, you're able to stream it. So as part of the page comes down, browsers by default already show stuff on the screen.

[00:54:03] JM: I mean, it's not just HTML though. You're also serving. It's like HTML and JavaScript, right?

[00:54:08] SW: HTML first.

[00:54:09] JM: Okay.

[00:54:11] SW: The way that pages load in the web is that you send the HTML first. HTML has references to JS and then does the subsequent request. So you want to have – If you want to have the maximum best theoretical performance, you have to have everything in the HTML. The

way you do that is user server side render. So this is a totally – It's a separate discipline from compiling. I'm trying to make that as extremely clear as possible.

[00:54:36] JM: Then when you're referring to compiling and you're saying like Vue is focused on compiling. What kind of compilation are we talking about here?

[00:54:43] SW: Compiling out its size. Everyone is sort of obsessed. To be fast, we have to have the minimum bundle size, right? If you want to provide features, you typically have to add to your bundle. So Vue, to have a default runtime – By the way, I don't know what the default Vue runtime is. Let's say it's 30 kilobytes. Let's say you only use 5 kilobytes of those.

Right now you're still shipping all 30, right? Which is extra deadweight. You're not using it. So the idea is that you're compiling – You statically look in your code and saying –

[00:55:11] JM: You send less of the Vue runtime.

[00:55:12] SW: Yeah, you're only sending 5.

[00:55:14] JM: Oh wow!

[00:55:15] SW: Yeah. That's a huge step change. It's huge for Angular. It's huge for Vue. It's just for Svelte. This is not a professional benchmark by any means. I previously had my blog on Gatsby. It had over 100 kilobytes of React.js code shipped by default. When I rewrote it to Svelte, it dropped to 9. So that's huge in terms of performance. It's hard to argue against that. Even though it will look fine on my house scores and it'll work fine on most modern American phones. That does matter for the rest of the world.

[00:55:44] JM: Sure. Yeah. You want your application to render performantly in Africa under like crappy cellphone conditions.

[00:55:51] SW: Right. Yeah.

[00:55:53] JM: I don't understand the word compilation here. You're using the word compilation to describe trying to reduce the amount of runtime. Meaning, like React has a runtime. Meaning like this is the way that React, the core React infrastructure interprets React components and allows them to perform as you want them to. If you can write your application in a certain way or if your application can be compiled in a certain way, you have to ship less runtime over the wire to the client. You're terming that "compilation"?

[00:56:30] SW: So you're essentially compiling the runtime. There's all the raw code for the runtime.

[00:56:36] JM: You're compiling your runtime based on the application requirement.

[00:56:40] SW: What you actually use. Yeah.

[00:56:40] JM: That's cool.

[00:56:41] SW: Stick that in your build tool and you got some pretty fast apps.

[00:56:46] JM: But React is not pursuing this idea.

[00:56:47] SW: React has explored ways to do it. Though the –

[00:56:50] JM: It sounds so hard.

[00:56:51] SW: By the way, this is like extremely sensitive. You'll get in Twitter wars of this stuff.

[00:56:56] JM: I can't wait.

[00:56:57] SW: Because people are very passionate about shipping performant code. Of course, that's our job. The way I put it is that React doesn't think this is the right thing to tackle. When you're compiling your framework runtime, which is your footprint of your framework. That's a local optimization versus the overall app code, which is hundreds of kilobytes.

Let's say Instagram.com is 2.5 megabytes. So what features can you stick – Let's say you take the runtime as a given. What features can you stick into that that optimize your overall app runtime? That's something that React is very, very interested in pursuing, because they have that problem at Facebook, right?

So they'll do things like code splitting with React lazy and suspense. They'll do things like partial hydration. So having a runtime, streaming things in and then hydrating things as you go along without blocking user input, because running that JavaScript does block user input unless you time slice it. All these React is basically the only player on the block doing that.

I kind of classify it as you either optimize your framework footprint, which is that 30 kilobytes, whatever, or you optimize the whole app, which you can also do in other frameworks, but React is trying to innovate in terms of the APIs to do that, to make it easier.

[00:58:14] JM: How did you get so fluent in programming in two years? You have a lot more expertise than the average person that started programming two years ago.

[00:58:26] SW: I don't know. The approach that I'm known for is called learning in public, and that's kind of like my hash tag that I try to encourage people to do. The main way to make sure you know what you know is to take notes of it, put it out there. Get corrected on it, right? To teach it to other people and you just be really, really freaking solid by the end of it. You cannot help but to be really good at it.

I think it just takes the guts to know that like, again, not everybody can do it, because some people are in vulnerable situations. But barring that, if you can put yourself out there, people come and support you. Like the way that I learned a lot about React is I started putting notes on about it. Danny [inaudible 00:59:08] came and helped me out on vetting some of the blog posts that I was doing, and I became better by that.

Then I got to practice messaging and got to make sure misinformation wasn't out there. Just by being a content creator and consistently creating, like sort of learning exhaust, you put yourself in the top 1% of like people. There's this – It's called the 1% rule. 90% of people don't – They're just kind of lurking. They don't really participate. 9% will comment and then 1% will actually

create stuff. By being a content creator and constantly putting yourself on the line and just reading everything available that's out there. We're talking about consuming information.

I went back into your backlog and listened to almost every episode of SEDaily, and I'm a better engineer for that. I think a lot of people, it's available to them. They just don't practice it in public as much. I will take something. I'll remix it. Put it in my own words. Put it out there. Let myself be corrected, because Paul Graham has this sort of philosophy of like keep your identity small.

[01:00:13] JM: Yeah, I love that one. That's a great essay.

[01:00:15] SW: My identity is not tied up in my work. I can be wrong, because this represents the best state of me right now. But please correct me. Please tell me I'm terrible at what I do. Because my biggest haters will be my biggest teachers. I think the more I sort of practice that, the more I sort of help me get better and learn from different perspectives, and it's been really helpful in my career.

[SPONSOR MESSAGE]

[01:00:45] JM: When you listen to Spotify, or read the New York Times, or order lunch on Grubhub, you get a pretty fantastic online experience. But that's not an easy thing to pull off, because behind-the-scenes, these businesses have to handle millions of visitors. They have to update their inventory or the latest news in an instant and ward off the many scary security threats of the internet.

How do they do it? They use Fastly. Fastly is an edge cloud platform that powers today's best brands so that their websites and apps are faster, safer and way more scalable. Whether you need to stream live events, handle Black Friday traffic, or simply provide a safe, reliable experience, Fastly can help.

Take it for a spin and try it for free for visiting fastly.com/sedaily. Everybody needs a cloud platform to help you scale your company. Everybody needs a CDN. Check it out by visiting fastly.com/sedaily.

[INTERVIEW CONTINUED]

[01:01:58] JM: I mean, I grew with that. If you can kind of subdue the most surface level ego drives, just a little bit. I mean, we all have our egos. You can't escape it. But like back when I used to play poker, I would always be asking people like, "Hey, I know I did something wrong in this hand. Please tell me what I did wrong. Please." Because you're not even like worried about feeling ashamed about this current situation. What you're worried about is the iteration and making the same mistake over and over and over and over again. You're just trying to avoid that.

[01:02:38] SW: Yeah. The way I put it is that you can learn so much on the internet for the low, low price of your ego.

[01:02:44] JM: Yes. I mean, I'm sure at the volume that you put stuff out there, you've probably been ashamed before. You're like, "Oh God! What did I do?" You're looking at an app that I made earlier today, and you commented on something about the appearance of it. I was like, "You know, you're actually right. This is abhorrent looking from the outside looking in." Good point.

[01:03:09] SW: But you shipped.

[01:03:10] JM: But I shipped.

[01:03:13] SW: Reid Hoffman will give you an A+ on that, because his whole thing is your first release. You should be embarrassed by it, right? If you wait till you're proud of it, you're probably too late.

[01:03:20] JM: That was like release 15 or 20 or something.

[01:03:24] SW: So it's a work in progress. I'm a tech speaker and I do a lot of talks. I'm not proud of the stuff that I do. The strange distributive nature of the internet, the virility of certain pieces of content means it's a maximum function and not an average function. That rewards

spiky people, rather than sort of medium, like stable people. You want to be unstable a little bit. You want to be spiky. Your worst stuff by definition –

[01:03:51] JM: Well, you don't want the downside to be too down.

[01:03:54] SW: Right. You shouldn't be a terrible human being. Your worst stuff by definition will not be seen and your best stuff will be shared endlessly.

[01:04:00] JM: Well, your worst stuff on Twitter will be seen and surfaced. You're talking about quality. Not like abhorrence of personality.

[01:04:09] SW: No. Yeah, that's a different dimension. Isn't it great? That just biases you towards creating more frequently often shipping.

[01:04:16] JM: I've made so much graph.

[01:04:18] SW: Yeah. Same here. I think JavaScript is a little bit like that. JavaScript has a ton of crap in it, a ton. But why does it keep winning?

[01:04:29] JM: Yeah, because the most people use it.

[01:04:31] SW: It's so easy to get started. You just get more people in it and people ship a ton of crap, but they make good stuff too.

[01:04:37] JM: Okay. A little bit of stuff on business. Indie hacker businesses versus venture-backed startups. In 10 years, will there be more people working for indie hacker businesses or working for large venture-backed corporations?

[01:04:52] SW: Wow! On that metric alone, I would say venture-backed. More people working for venture-backed, and that's just because indie hackers have their own very particular unique mentality, and a lot of indie hackers are drawn because they don't want to work with a ton of bureaucracy and code. Just like in terms of raw numbers, corporations are made to scale. We have the playbook for that. Indie hackers, each individual indie hacker needs that activation

energy to be the creator and just like a lot more people are not in that game, if that makes sense.

[01:05:29] JM: 10 years is not enough for a cultural shift in that direction.

[01:05:32] SW: I don't think we'll ever get there. It's just – We're talking about – We just talked about Zipf's law, the 10%, 9%, 1% rule. To be an indie hacker, you have to beat the top 1% of the 1% actually to ship and take the instability and go do something on your own. It's so much easier to be middle management. It's so much easier to have status at Google.

[01:05:51] JM: Easier than the short-term.

[01:05:52] SW: I mean, you do decently well at a big FANG company right now. So it's absolutely not clear.

[01:06:00] JM: Yeah, not clear. Give me your take on influencer engineers.

[01:06:05] SW: Ooh! They exist. You cannot deny – A lot of people try to pretend that they don't. I will say this. No Netlify, Stripe, freaking Apple. There is the regular support line and then there's the influencer support line. Nobody will fess up to like treating influencers differently than regular people, but they do, right? That's what DHH did with the Apple card instant recently.

[01:06:39] JM: I don't know that story.

[01:06:40] SW: Oh! Do you want me to tell you?

[01:06:44] JM: I assume basically the story is DHH, he's a loud voice. Somehow got to the front of the line on some issue.

[01:06:52] SW: That's fine. It happens in consumer services. You're a big deal there. But also in developers and engineers. That's a reality that people pretend doesn't exist. We want to treat everyone fair and equal. Unfortunately, people have – Some people have louder voices than

others and they do get preferential treatment. Just as a result of normal human behavior, not necessarily a result of any sort of broad bias.

So David Parel who studies these kinds of things calls these naked brands. The idea that we want to achieve more authenticity that we want to identify with individual people rather than overall companies and influencer engineers are a bit culture of that in software I think. Who are the brand names in cloud? I think of Kelsey Hightower and anything that he likes, I automatically like.

We want to delegate to authority. We're just like kind of tribe or herd animals like that. That's absolutely fine, because people have skin in the game. Influencer engineers know that like if they becomes corporate shells, they'll be found out and all the sort of hard work that they had will be lost. So they try very hard to keep their own personal brand separate from the company brand. That benefits the company as well as the engineer. Arguably, I'm on that path.

[01:08:18] JM: Somewhere.

[01:08:19] SW: And I'm not super comfortable with that. I worry that the mob will come for me.

[01:08:23] JM: Dude, for sure. In this day and age, for sure.

[01:08:29] SW: Recently, the former manager of React, Sophie Alpert, she recently got an email saying that they would pay her $600 to contribute to a GitHub repo just to give that veneer of credibility. A GitHub repo, right?

[01:08:46] JM: Dude, you should see – I mean, the crypto. The crypto stuff is way worse. Crypto influencer.

[01:08:51] SW: Oh, yeah. There's a direct financial sort of incentive there. Whereas for code, it's not so clear. It's mostly about influencing long-term purchasing decisions.

[01:09:03] JM: But that's hilarious. That is the $600 for Sophie Alpert to commit to her pository. It's a naked influencer stuff.

[01:09:10] SW: It's real. So marketing finds its way around in different forms. Your podcast, you have a ton of influencer engineers come on, and that is absolutely the way we've rather be marketed to. My team is called developer experience, and what? There's this whole trend of developer advocates and evangelists and experience engineers coming up and we just don't want to be sold to by someone in a marketing department. So we should just call it a different name.

[01:09:38] JM: Yup. Pretty much.

[01:09:39] SW: I should disclaim, because I don't want to get fired for this. That's a cynical view of this. The more informed view is that it's a two-way street rather than a one-way street.

[01:09:45] JM: Oh, definitely a two-way street. I mean, it's just like a better form of marketing than airport ads. If you're going to spend your money on something, don't spend it on airport ads. Spend it –

[01:09:54] SW: There's this joke going around that no one ever got fired for buying an airport ad, and that's why you still see so many in SFO. So influencer engineers are in the rise. David Perel calls these naked brands. That's a very important insight that I think everyone should be aware, is becoming a factor in our environment, in our economy. I don't think it's necessarily unhealthy. I think people need to be authentic. They need to be sort of held to account if they are bad actors. But on the whole, more skin in the game is always good.

[01:10:25] JM: Okay. Last question. Any meta advice for how to ramp up quickly as a developer? Somebody who's at the beginning of their career?

[01:10:34] SW: I've done a couple of talks and an essay on this called Learn in Public, and that's the meta advice, that like you take what you learn and you put it out there, because a lot of people learn in private, which means they take what they learn and they sit on it and forget it. They maybe apply it at work and then that's it.

But like if you want that magnifying, like focusing glass of public attention on your work to make

you the best that you could be, that's a really good sort of meta advice to accelerating. So once you start doing that, once you start making little, tiny little things, like blog post and meet up talks and all that, then you graduate towards the bigger things, like courses and reusable resources. One of my best projects a React TypeScript, is basically the de facto of React TypeScript documentation for the community, and tons of people are adapting React and TypeScript, and I've taught all these people at Uber, Atlassian, Pinterest like React and TypeScript. That is just for me creating these sort of reusable resources.

Now, first of all, I'm a thing. So therefore the TypeScript community has to go through and vet my work, and that's great, because I get taught by the TypeScript team. Now when every new piece of information that gets PR'd in, I learn from that as well. It's kind of like you know you can open source code and your code gets better maybe by peer review. You can open source knowledge as well.

[01:11:49] JM: Totally.

[01:11:50] SW: And maybe your knowledge gets better. I think the primary function you provide there is writing and organizing information for people like you, and there may be a lot of people out there, but there's definitely a tribe of people like you out there. They're looking for you. They can't find you. So just put out your work.

[01:12:07] JM: Shawn Wang, thank you for coming on the show. It's been a pleasure getting to know you over the years.

[01:12:10] SW: Thank you. Always a pleasure hanging out.

[01:12:11] JM: Glad to meet you on the internet.

[01:12:13] SW: Same. Likewise.

[01:12:14] JM: We'll keep the conversation going.

[01:12:16] SW: Cool. It's a pleasure. Thank you for having me.

[END OF INTERVIEW]

[01:12:26] JM: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[END]