**EPISODE 927**

[INTRODUCTION]

**[00:00:00] JM**: Webflow is a platform for building applications without programming. Software engineering has been around for barely 30 years. Over that period of time, there have been many attempts to create a platform that allows for the creation of software without writing a line of code. Most of these systems have not been able to fulfill that task, and this should come as no surprise. It's hard to build an application even if you know how to program.

Vlad Magdalin has been working on Webflow for more than seven years. He has persisted through multiple failed attempts at building Webflow and he's pushed past continuous rejection from investors who did not see the viability of his vision. As Vlad patiently worked on Webflow with his two cofounders, the power of the web browser slowly improved. V8 became a powerful runtime that could deliver the performance necessary to build applications visually in the browser.

The unmet goals of past WYSIWYG application platforms faded into irrelevance as Webflow came into being itself and allowed for an entirely new type of software development driven by the visual interface of the browser. Webflow is one of the coolest, most ambitious software platforms in existence. Vlad joins the show to talk about Webflow and the future of software.

We're hiring a head of growth at Software Engineering Daily. If you like Software Engineering Daily and consider yourself competent in sales, marketing and strategy, send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

**[00:01:45] JM**: I never liked searching for a job. It's painful. Engineers don't want to make a sacrifice of their time to do phone screens, and whiteboard problems, and take-home projects. Everyone knows that software hiring is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with more than 400 tech companies including Dropbox, Adobe, Coursera and Cruise. If you've been hearing about Triplebyte for a while, you'll be happy to know that Triplebyte just launched a brand-new machine learning track and they'll now be helping machine learning engineers find jobs in the same way that they've already helped generalist, and frontend, and mobile engineers. It's amazing seeing Triplebyte expand into these specific verticals because they're so efficient about matching high-quality engineers to great jobs.

Go to triplebyte.com/sedaily to find out more about how Triplebyte works. You can take a quiz to get started, and if you end up taking a job with Triplebyte, you get an additional $1,000 signing bonus because you used the link triplebyte.com/sedaily. If you make it through that quiz, you get interviewed by Triplebyte and you get to go straight to multiple on-site interviews.

It's economies of scale for software engineering interviews. It's is pretty sweet to see that centralized in a place that gives you those economies of scale. I'm a fan of Triplebyte. I hope it gets bigger and bigger and creates more and more economies of scale in the miserable hiring process of getting a job as a software engineer. Make that painful process a little bit better with triplebyte.com/sedaily.

Thank you to Triplebyte for being a sponsor.

[INTERVIEW]

**[00:03:58] JM**: Vlad, welcome to Software Engineering Daily.

**[00:04:00] VM**: Thank you.

**[00:04:00] JM**: You work on Webflow, which is quite amazing. Describe the domain of low code or no code tools.

**[00:04:11] VM**: We like to think that we play more in the no code space and that the way that I see the gradient is like full code is the way we build software, like Ruby on Rails, or JavaScript,

or React. That's like you have a text editor open or an IDE and you're creating software from scratch. That's sort of full code or just code.

Low code is when you start to augment that with visual kind of process automation or visual workflows that start to replace some of those code aspects. One example I can give is let's say a marketing automation software. You can say, "Oh, when somebody clicks on submit this form, send them an email two days later." Typically, that stuff would've been done via code pattern 10 years ago. Now we have brought in more things that can kind of express that logic, that business logic in a more visual environment that more people have access to.

But low code companies typically tend to be in the upmarket enterprise space where they're trying to do what's called like shadow IT and like entire application replacement, which is not possible to fully do it without code. So what they'll do is they'll say, "Here are the things that we've already solved with like visual tools," let's say workflow automation, when this happens, something else happens. But they'll still rely on coders to do certain things that that sort of framework doesn't do.

No code is like full on you're never writing or the goal is to try to replace everything that you do encode with a declarative visual. It doesn't have to be visual per se, but some sort of non-textual representation of like declaring logic, or UI, or a database, or a schema, or something like that.

Examples of that are – I mean, even take Webflow. Instead of designing a database or saying like, "Oh, I'm going to add a column or add a row, you were in like MySQL, or Mongo, or whatever, those things are done either through the command line or through some sort of like admin interface for your kind of going in and creating, like writing a query or writing a statement to like add an index or things like that. You're literally going into UI and saying, "Add a collection, and add a field to the collection," and all of those sort of technical details are taken care of behind-the-scenes.

Also a Webflow example, instead of writing HTML and CSS in two different or in a text editor, you're visually manipulating HTML and CSS driven layout, but you're doing it through a drag-and-drop visual interface. It's like a different way to think about like putting together code. Maybe a concrete example is instead of writing div, a paragraph in HTML, you would write like

<p> tag and inside the text that you want that tag to have, and then the closing <p> tag, that's code.

In a no code environment, you would drag in a paragraph and you would like physically – Like you would see that on the screen. You would double-click on it. You would change the text. You would maybe instead of like putting bold tags or italic tag, you would just select some text that you want to be bold or italic and you just click a button. So that's like of fully no code way of doing something that traditionally was done with code.

The ultimate goal of all these things is just to create more software. How do you lower the complexity curve to get people to create the same kinds of things that software engineers are but to do that in a language that is a lot more approachable to a lot more people?

**[00:07:41] JM**: If I'm a software engineer, should I care about this stuff today or just keep my eye on it?

**[00:07:46] VM**: You should keep your eye on it. A lot of software engineers care about this stuff already, especially when. So for example, there's a lot of software engineers that use Webflow, for example, or even things like Zapier for quick automation of things that they just like can't be bothered with.

They know it's much – For example, let's say I wanted to do something like when somebody post a tweet, I want to get a text message. I can spin up a node server. I can spin up like some sort of like background cron job to keep checking and then like pulling the Twilio API and pulling like the Twitter API and do all these glue, or I can literally just like go to Zapier and say, "Connect my Twitter account. Connect my Twilio account. When this happens, do this to this phone number." Boom!

Even though you know how to do that, it's much faster to do that. You can think of one parallel to this the spreadsheets. Software engineers used to be like doing all the work that spreadsheets currently solve like 50, 60 years ago. Financial modeling was done through like a COBOL or Fortran, a Pascal developer. A business user would say like, "These are the calculations that we need to do, that one trivial that you can do on a calculator, and we need to

like generate some graphs and some insights from the state." A programmer would actually go implement all that in code and get sort of a graph out or whatever.

Spreadsheets completely, for the most part, remove the need to do that. So like software engineers were able to move on to more interesting problems because that subset of computing is now solved by like a much better, a more approachable kind of interface to solve the same problem. In that same way, a lot of software engineers are already using Webflow to, for example, build HTML and CSS layouts. It's just fundamentally faster.

**[00:09:26] JM**: Can you export it once you –

**[00:09:28] VM**: You can. Yeah. It's basically an abstraction over HTML and CSS. It's just like dev tools. You know how a lot of developers will go into dev tools and they'll use like the dev tools color picker instead of like writing a code by hand, writing a hex code by hand. They'll use the color picker or they'll use like the little Bezier editor. But the key difference there is like once that code is done, you have to sort of copy-paste it back to your source. Webflow is just that on steroids, where it not only saves directly back to the source, but also adds a lot more visual tooling to take those, kind of like act on those same abstractions.

**[00:10:02] JM**: How do you get to the place where it's one click export to React components?

**[00:10:07] VM**: I think that's actually, believe it or not, shorter-term thinking. React is an implementation detail, just like HTML and CSS is an implementation detail. What we're working towards is the ability for designers and design teams to create the entire design system. Just like InDesign, if you think about print design, et cetera –

**[00:10:27] JM**: I think most of the listeners have no idea what InDesign is or –

**[00:10:28] VM**: Okay. InDesign is the way that magazines are made.

**[00:10:30] JM**: Or even what's a design system.

**[00:10:32] VM**: Okay. A design system – Actually, let me scooch back a little bit like the way that we do design is think of it like magazines and newspapers, etc. That used to involve a lot of programmers. It was people, designers pasting stuff on like – It was literally called paste up. You do it with glue and like scissors, etc. We had the skill called print setters that would sort of translate that to PostScript that would go to print.

**[00:10:58] JM**: By the way, you know this because you went to art school, right?

**[00:11:00] VM**: Yeah. Yeah, here just down the street, and I was designing magazines, etc. But then digital publishing software like InDesign, QuarkXPress, etc., they sort of replaced that need to just do all that stuff by hand.

When you'd create a magazine now, you create what's called a sort of a design system, which is all your topography, all your spacing, how your grids work, etc. All your sort of like common elements and tools like InDesign let you create that for magazines. Similarly, web design sort of got inspiration from that and started creating design systems for websites and web applications.

Basically, what that means is like you find common patterns between the website you want to create or the product that you want to create. Instead of repeating that all the time, let's say you have a user profile like circle with a shadow that has like the person's name under it and that appears in the list of users, in your login view, on the blog post, like byline. You sort of use this like component everywhere and you like pull those out into separate components, and that creates kind of like your set of building blocks that you build with.

React came out from that concept of like how do you create these components that serve as building blocks that you can then compose with to various levels of complexity. So React is very much like the evolution of that, of like creating a design system, but that's still an implementation detail. The hard part is still how do you actually – Which components do you need, to how do you like to minimize the number of like divergence you have in those components? You have like a unified system across like whatever software that you're building.

That's why all these teams like Atlassian, and Airbnb, and Netflix have pretty big design systems teams that are creating the building blocks that the rest of the teams, the software development

teams are using as kind of like the basis of how they put together UIs and sort of the experience of how people like flow, like the customer journey through that software.

When we're talking about – Right now, the way that design systems are created as a designer like declares what they look like in a tool like Sketch, or Figma, or Photoshop, or whatever, and then a developer builds the design system in either HTML,  or CSS, or Reactor, or Vue, and there's a bunch of tools around this stuff, like React has things like Storybook where you'll put all of your components into one place where other developers can go see and like, "Oh! Well, we have a drop down, and we have a slider, and we have an accordion, and we have this user avatar, and we have a media card component." So they know what the building blocks are before trying to figure out if they need to build their own.

But the key critical part is that there's this translation layer. You go from a design file to code and you're not really quite sure where the source of truth is. You're not sure, "Oh! Some React developer went and changed the media card component to make the picture bigger and the heading smaller, because it looks better or whatever."

But in the design file, it still looks like the previous design. You kind of have like this – You don't know whether the code is the truth or the design is the source of truth, because then you go to design like the next feature or the next iteration of that feature. The designer will go back to their design file and what they have is actually no longer a reflection of the actual application, and you have this weird divergent.

**[00:14:19] JM**: They solve that.

**[00:14:20] VM**: That's called the designer developer handoff problem right now, and it's a huge problem. Designers claim Figma is the source of truth, and developers are like being lazy by not following exactly what the designer said should be what it should look like. Developers say React is a source of truth.

**[00:14:36] JM**: That was like a dev ops problem.

**[00:14:37] VM**: Yeah. I mean, there're a bunch of problems there. But what we're trying to do is to create the source of truth as a not necessarily in the design tool nor in the development tool, but in sort of like this middle layer where it's almost like a visual implementation tool. That's why we call Webflow like a visual software development platform, not a design tool and not like an IDE or an integrated development environment. That buys you the ability to say that designers can now create entire design systems, but because of the way that they created them, it's already in production. They're doing it visually, but the actual components no longer need translation.

So you can have a developer not even export the code, but literally import like via a URL or something, that component itself. All they really have to care about is the API of the component, like the props that it takes, etc., but the design is driven by the design system, which is driven by the design team. That removes that need to go between like what is the source of truth. It ends up being just like if you're doing – If you're creating a logo, that source of truth is in Figma, or Illustrator, or Sketch.

**[00:15:44] JM**: Hearing you explain this, actually, it sounds like we have the tech, the core technology to build this today. That kind of thing could exist if we had like – You get the right web assembly engineers in the same room with the all the building blocks there. You can get this thing going at some point.

**[00:16:00] VM**: Right, but it's mostly not a technical challenge. It's actually more of a human challenge, because you have these kind of silos of certain things are only possible to developers, and developers – I'm a developer. We like having this kind of feeling of this stuff is only possible because we're here.

**[00:16:19] JM**: This is one thing I've been like really thinking about, and I've been chided by some of the listeners who are telling me that like I'm getting – Like caught in the hype and I'm like, "No. I think actually there's something significant that's going to happen here," where like developers are doing a lot of unnecessary work and designers are probably not working as efficiently as they could. It's just very hard. I mean, it's very hard to keep up with all the tools and it's hard to know what's possible at any given time. Yeah, it's like an adaption question. When do

people adapt this stuff and what are the team roles that change? We need different team descriptions almost.

**[00:16:55] VM**: Yeah, and we're coming up with some of those, what we call like a visual developer, a person that's more like design and UX-centric.

**[00:17:03] JM**: Visual developer.

**[00:17:04] VM**: But they learn the core – Not the syntax, but they learn the fundamental concepts of web development. Things like the box model. So you don't think of like dragging something and like absolute positioning. You're actually thinking of margin padding, fixed positioning. You're thinking of kind of the box model as everything is a box. You're not just moving things around on a fixed canvas. You're thinking of things in relative terms just the same way a CSS developer would think about it. You're not actually the writing code. You're understanding those core concepts and having visual tooling that lets you take advantage of those.

One example is, back in the day, when we had graphic design, people would create logos. When SVG or just vector graphics in general were becoming a thing, we didn't have tooling like Illustrator. What happen is illustrators, the people responsible for making a logo, let's say the FedEx logo or whatever, 20 years ago. Much longer than that actually, like 30 years ago. They would actually do it on paper. That was graphic design. You literally did it on paper.

Then a postscript person would go and create like all the curves, like these Bezier curves, because they were familiar with like the math and how you define the stuff in PostScript, and that then goes as like the super high-fidelity source of truth that goes to the printer, right? That goes to like the high-quantity run or whatever. Those developers are like pretty protective of like, "Okay, we know how to do this. We know how to do like declare gradients and like Bezier curves, etc."

But over time, the source of truth moved to – The role of the graphic designer expanded to where they were now responsible for learning some of these digital design tools, like Illustrator,

where they were now empowered through the power abstraction create that final source of truth, that PostScript file, or that AI file, or that SVG file, and tools generated that now.

So it's almost like the role of a graphic designer absorbed more responsibility. So now whenever you say, “Oh, we need to do a logo update.” You literally go back to your graphic designer and they send you the file that is like the source of truth for that update. It doesn't have to go through a translation layer.

Similarly, I think product designers, especially private designers working in software and web or like web designers, their role will expand to say, “Now, you're not just creating a flat design file, like an idea of what you want to build. Now you have to learn more about the medium that you're actually creating for,” and those are things like instead of saying creating a file that has redlines that says, “Here's a blue button, and when I hover over it, it turns red with like a little annotation that says basically instructions to a developer that say, “Turn red when this is over with this sort of like quarter second transition, with the ease and out sort of like timing curve.”

You’re actually working in software that helps you declare all that in the native medium of the web. So you’re creating that button, choosing the color, switching to like a hover state, changing the color to red from a color picker, then going back to the original button in the non-hover state and saying like, “This will have a transition for a background color over this time period.”

So that person is now basically doing the translation work that a programmer would've done, but it means that that translation step is no longer necessary. This is kind of like it sounds kind of threatening to programmers because it’s saying like something that we do right now is being taken away or like moving into like our services are no longer needed. But what it actually does is it empowers programmers to work on like much more interesting things that just like have no chance of being automated anytime in the future.

Just like those PostScript developers, they moved on to working on game engines or working on like other things that just require much more –

**[00:20:45] JM**: Are we over this as developers? Haven’t we gotten over this, the fear being outmoded? Nobody actually fears that anymore.

**[00:20:50] VM**: It's seven years ago when we're getting started, that fear was huge. Right now it's still. I would say it's at 80%. The vast majority of WordPress developers, and developers were not quite – The vast majority of the job they do is translation from a design file to like a WordPress theme or kind of like making it real, bringing it into production. I think there's still a lot of that happening. Most of the work comes through like agency where there's a design team and the programming team is the one that's implementing the designs.

There is like from a lot of the conversations that I've had, like people still feel threatened. I think programmers need help in seeing the opportunity of like how many other companies need like this sort of low-level programming skill. We're literally facing a 1 million developer shortage in machine learning, in AI development.

**[00:21:44] JM**: I mean, that's the understatement of the year.

**[00:21:46] VM**: Yeah. But I think, naturally, we as humans, we don't want to grow outside of our comfort zone. When I was a developer and I was using Knockout in our team, building Weblow with Knockout, I was like super comfortable with it. When our team proposed that, "Hey, this probably isn't going to last that long. Let's switch to React." I had this feeling of like, "Oh! Maybe my skills aren't going to be that valued and I'm going to have to learn something new. I really don't understand this new mental model." It was like it took a while for me to get on board with. So I totally understand that, that fear and that pain. But I do think that, ultimately, it's bad for the world, like in aggregate, to avoid efficiencies that could be just much better.

When spreadsheets became the de facto way of doing financial modeling, when you have over a billion users of spreadsheets on like a monthly basis, right? Whereas we only have 20 million programmers. If we still had the limitation of you had to do the kinds of things that spreadsheets enable you to do to have a programmer, we wouldn't have entire companies created. We wouldn't have like people – I just had a plumber come over to my house and he had like this whole like Excel model for doing estimates. It was genius. He could click something and it changed something else in a different sheet and it generator the invoice. It was amazing. That business would be less efficient, because there's no way like this plumber would hire a full-time programmer.

So think developers have to think of it from that point of view of like how do we get more of the world capturing this ability to make software and benefit from it. Because right now, literally, 0.25% of the world is creating software. Most of us are just consuming it, because only one out of every 400 people is a software developer, like they can actually create software. Sometimes I make this analogy of, “What if everybody could read?” Kind of like what we have right now in terms of literacy rates and everybody is able to access the Internet and like use the software that we create. But what if only one out of every 400 people was empowered to write, like a story, or a book, or whatever? That’s the disparity of where we are today right now in terms of software creation and software consumption.

Imagine if only one out of every 400 people is writing books. That's where we were 400 years ago, like pre-Renaissance. When the printing press was like still very closely held to the chest and like governments sort of saw it as a threat that people would learn how to like spread information via written form. I think we’re still in sort of like day one of the Internet type of timeframe when it comes to creating the kinds of things that can take advantage of the kind of connected software and services and products the people can create.

[SPONSOR MESSAGE]

**[00:24:30] JM**: As a programmer, you thinking object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud era. Millions of developers use MongoDB to power the world's most innovative products and services; from cryptocurrency, to online gaming, IoT and more.

Try MongoDB 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.

[INTERVIEW CONTINUED]

**[00:25:25] JM**: I want to talk about the engineering behind what you've built, because I've used Webflow. There is so much stuff in the interface, but it's performant. That's a rarity. I mean, it's not easy to build a beautiful, highly interactive single page web application that is performant.

**[00:25:48] VM**: In a single thread.

**[00:25:49] JM**: Yes, exactly. I mean, that's JavaScript, right? JavaScript has to be single threaded, and it basically feels as response – I mean, maybe it's an exaggeration, but it feels almost as responsive as like an IDE on your computer, or Sketch. How have you kept the performance so high?

**[00:26:08] VM**: To be fair, it's not always that high, especially as you start to build really huge sites. There's a lot we're working on. Honestly, it's all constrained by what a browser can do. We definitely pushed the browser to the limit. I think we have one of the most complex in-browser applications in the world like in terms of trying to essentially build an application inside of a browser. But a lot of the performance improvements have come from – Surely, there's a ton of kind of optimizations that we make on our team, but also browsers in general are getting way more performant. The V8 kind of VM for Chrome, and Safari, and Firefox is now getting faster and faster.

So there's sort of the combination of that of like computers in general feeling faster. Webflow doesn't feel that fast in a really old Chromebook. So there's definitely more that we can do. But for us, it's just the limitation of like we had to build it in the browser, because our core constraint is just like dev tools, or Web Inspector when you open it in Chrome. That only works when you're working on the real thing. It's a true WYSIWYG. It's not a design tool that then gets translated to code. It's essentially dev tools with a cleaner UI. You have to work with the real DOM nodes. You have to work with the real like CSS. When you change something, you're actually changing the CSS via the CSS object model. So it's not like an approximation of what it's going to look like. It's the real thing. You're actually like direct manipulating that thing.

So we had no choice and to just kind of going in that direction, which is why think a lot of previous things didn't really work, like iWeb was like an old Apple kind of website builder. They had to build their own sort of like emulation for what a browser renders. But the fact that we run

directly in the browser and use the browser's engine and the actual canvas is just an iframe. It's the actual website that's going to get published. That I think we just get the tailwinds of as browsers get faster at rendering webpages, so do our tools. They get faster because of efficiencies gained through websites getting faster in general.

**[00:28:14] JM**: How has the frontend changed overtime? You said, for example, you moved from Knockout to React. Was there a point at which you refactored everything to use React?

**[00:28:28] VM**: Basically, yup. Knockout was something that when we were first starting to build Webflow in 2012, Angular was already around. JQuery was around. BlackBone was around, and Angular was around, and we tried all of them and they were all too slow except for Knockout.

About – I think it was 2015, we saw just like the entire community around Knockout just kind of die off. I think the main maintainer got a job somewhere else and stopped maintaining it and we were the only major app that was built in Knockout outside of something internal at Microsoft, which was sort of like where the main author went, I believe. So it was sort of the writing was on the wall and we had a really hard time hiring people, because nobody knew Knockout. Nobody wanted to work with Knockout.

At that point, like React was starting to become popular, and like the mental model of React was more appealing even though it's something new that needed to be learned. It was something that a lot of developers really gravitated to. So we almost, I would say, had no – I think it's good that we were early enough. Had that exact situation happen now, let's say we were still built on Knockout at our scale now, it would've been much harder to switch. Almost like Facebook was sort of – Even though they saw PHP going out-of-favor overtime, they had to kind of like – They had so much stuff built on it that they essentially –

**[00:29:49] JM**: Let's double down.

**[00:29:49] VM**: Yeah, they doubled down and they invented their own like PHP virtual machine or whatever to make it faster. They just didn't have the option to go switch as unappealing as PHP was sort of like culturally or whatever among developers. So we spent a little bit less than

a year, I think, to just do a pretty much [inaudible 00:30:08]. That has been a major benefit to us overtime.

**[00:30:14] JM**: I interviewed Howie from Airtable, and they actually built their own –

**[00:30:20] VM**: Database and –

**[00:30:22] JM**: Well, database, but also like JavaScript frontend thing. They don't use React, like some custom system. I mean, are there any performance things where there's some like low-level defaults in React that are giving you performance hits where you're –

**[00:30:36] VM**: Oh, there's a bunch of times we have to eject from React or from other types of tools because we have to sort of create a hyper performant version. One example is – This isn't React, but there's another kind of engine that helps take like some CSS and turn it into – Like move that, some CSS changes, and make those changes in the style sheet.

So we had to invent our own essentially like reconciliation engine that says, "Here's like the old CSS. Here's the new CSS. Figure out just like the core differences between the two and make those like micro changes via CSS OM so that you're not like causing thrash in the thread and like getting the browser to like have to re-render too much.

So there's a bunch of times that we have to do that, and there's a lot of examples where I believe – I forgot what it was called, like Adam, the editor that GitHub built, where they're using React but then for like very fine-grained things, like text editing selection. They had to like invent their own like hyper performant version. Just because like there's – React is kind of – React is taking a lot of steps now with like 16 and they're working on sort of async. They're working on some really sophisticated things to improve performance and give like fine-grained control over like which things get deferred and which things render kind of in the next frame.

But before that was available or as that becomes more mature, you just have like very specific problems that just the cookie-cutter approach of React just doesn't work. It is the simplest one, because you have like the simple mental model of like, "Oh, whenever sort of state or props change, I just like trust that the reconciler will figure out the differences and render, just make

those small changes in the DOM." But sometimes the reconciliation process itself is very expensive in order to figure out what the differences are. But if you know what the differences are, you can just go make them directly, which is sort of like the old school way that we used to do things in like jQuery and whatever. Developers had to keep track of all these things. I know this is changing the UI, then go change this.

**[00:32:36] JM**: Wasn't that fun?

**[00:32:36] VM**: Yeah. There's quite a few times that you needed to eject, but that's software engineering. You have to figure out where the bottlenecks are and kind of – Yeah, in some cases, reinvent the wheel. Well, not in a bad sense, but you have to like solve that problem in a way that's specific to your content.

**[00:32:53] JM**: That's what React native developers do all day.

**[]00:32:55 VM**: That's true.

**[00:32:56] JM**: What's your backend database? What's your source of truth?

**[00:32:59] VM**: We have node is the primary one. Sorry, not node. Mongo. We have quite a few. The source of truth is actually JSON. So all Webflow applications are actually a graph of different JSON primitives, so like a node or like a div for example is a primitive. CSS class, we call it a style block, is a primitive. You actually won't have things like div class equals some class name, and then there's something in a style sheet. There's actually JSON primitive that says, "Here's the ID of the style block and all of its CSS properties," and all of those are like a much lower level than CSS. They'll have like tokens for like color values that are specified by a design system or by a designer, then that all gets compiled into HTML and CSS.

But what that allows you to do is enables you to create a graph of dependencies. So as a developer, you'll never go into an HTML file and give every single node an ID, because it's just like looks noisy, etc. But when all that stuff is behind the scenes in JSON, it actually gives you a lot of power, because what developers have to do in their minds around like, "Here's my

component before. I had like this element and I added this wrapper and I added this class and maybe I changed the style."

Then they have the old version. What they do is they reconcile those things in their minds as they do like get merged, or if there's a conflict they have to say like, "Oh, I made this change, etc." But when you're doing everything in JSON and you're doing it atomically, you know exactly which change was made to which element and you treat those as just like operations in a database and you know what those links are between things – If you try to delete a class or a style block that has certain styles, you know exactly which nodes connect to it.

You know when you move one div from one place to another, you know exactly which one you moved. It doesn't just look to the editor as like, "Here are some new code that gets added in this place." So it knows exactly the history and context of what that node was and all the data associated with it.

So it lets you do things that you basically build an in-memory application graph that says, "Here are –" And it goes as deep as like data dependencies. So a lot of times, developers keep this in their minds. They say, "Oh, I know the schema of my backend has like these fields, or these columns and this database table."

**[00:35:15] JM**: You have basically a backend in-memory representation and you snapshot that into the database on a regular basis, but then your frontend is just reading from this in-memory thing.

**[00:35:25] VM**: Correct. Yes.

**[00:35:27] JM**: That in-memory thing, you just have some cool backend model that's constantly like figuring out when to snapshot it.

**[00:35:36] VM**: Pretty much, yeah.

**[00:35:38] JM**: That's cool.

**[00:35:39] VM**: The hard part is getting that to be like real-time, because you might have somebody working –

**[00:35:43] JM**: Yes, real-time.

**[00:35:45] VM**: Yeah, one a person working on one component on the homepage, because they're doing a design. Then somebody else on like the about page.

**[00:35:52] JM**: Oh no! Your Google Docs.

**[00:35:53] VM**: Yeah, exactly.

**[00:35:54] JM**: I forgot about that.

**[00:35:55] VM**: But it's even more complicated than Google Docs, because Google Docs has the advantage of sort of like doing text comparison. So you have like the context of what intent is with when you're working on like a style sheet that's shared across many different pages and dynamic context. You kind of have a lot more like propensity for conflicts, because somebody might be changing a color based on some class, like some button class in a dynamic list of blog posts and they think it looks good there.

But that same style is used in a list of a blog post authors or something like that, or maybe like a list of investments, and they might be making a change that is sort of conflicting, like doing a different width, or padding, or whatever. So you have to like find ways to reconcile those things and like surface conflicts in real-time to sort of help designers make a choice of which one is the correct one.

So there's a bunch of problems that we still need to solve there around essentially having to abstract away like this act of like committing code into GitHub when you start – Or merging code, where you have to start dealing with conflicts when – That are happening manually right now to like code review, but we have to like get them to happen kind of like live in real-time.

**[00:37:10] JM**: That sounds not fun to implement.

**[00:37:11] VM**: Yeah. But it's an important problem to solve.

**[00:37:14] JM**: Totally. So you can do like GraphQL stuff too, right? So you have a CMS. So if I build a data model in your CMS, there's like a way that GraphQL aligns it, right?

**[00:37:27] VM**: Yes. So we essentially abstract away GraphQL, like it's involved but you just don't know it. Our CMS is essentially like a visual database builder. So you can build relationships like one-to-many relationships, many-to-many. You can say, let's say, I wanted to build your podcast website and you have an object or a collection called guests, and you have a collection called topics, and you have a collection called episodes, and you have a collection call companies, or whatever, and you can start to build relationship between those things. You can say a guest is related to a company and they were on this episode, and this episode has like all these properties of like date, etc., like audiophile linked to iTunes or whatever.

So then once you build that, it essentially is a – You create a database schema. From that you can get like a GraphQL endpoint. So you can just use GraphQL to inspect that data and get data out, but that's not the most interesting part, because there's like other kind of GraphQL abstractions that do that. I forget their names, but like Prisma, I think.

But the important thing is that now in our visual UI tools, you can actually build around that dynamic data. So you drag in something, we call it a collection list, but it's essentially a repeater that's bound to data. At first it's empty. So it has like it's almost like an empty GraphQL query. But then you start dropping elements into it. Like let's say I drop an image, and that collection list knows the context of the collection that I'm iterating over. Let's say I picked podcast episodes.

So it will know the available fields that I've defined before that are relevant episodes. So it will say title, guest, etc. A guest is a reference. So it's essentially in SQL terms like foreign key. As you start to build that UI, we're automatically building a dynamic GraphQL query that says, "As you're binding –" Behind-the-scenes, we're writing a GraphQL query or essentially the contract of what this component expects to pass down into its props from the backend data source.

The designer, when they're working with Webflow, they don't actually know that they're creating a GraphQL query as a design, and it sort of like writes itself as we figure out, “Okay. So you only want the podcast like thumbnail and the name and a link to it's like detail page.” Therefore, you never have to load the guest, or the guest avatar. But then if you drop another image and you say, “This image gets its source from like the podcast guest’s avatar picture,” like my picture for example, then it automatically does that as like a nested part of the GraphQL query that says like, “Now I want to load the guest.” For the guest, I just want to load the avatar image and maybe the name if I want to have that as like a label right next to.

So it’s almost like it's doing the same thing that a developer would do with like defining the backend data model, and then writing the GraphQL query, then writing a React component. With Apollo, that’s somehow like co-locates the GraphQL query with the component, but it's all doing it behind-the-scenes and kind of abstract it away. None of our – We call them visual developers. They don't need to know the implementation details. We can actually swap out Apollo, which we've done in the past, to a different provider. As long as the user experience is the same and the output is the same, that dynamic data is now loaded on the canvas and then we’ll like actually reflect on the published page. It doesn’t even matter to people building their like sites and experiences, like how that's all implement. It's an implementation detail.

**[00:40:49] JM**: Pretty sweet. Do you use TypeScript?

**[00:40:51] VM**: Not yet. We use Flow. So that was for us a more natural way to get types that didn't require like a full-on change to a different language. I know there’s like ups and downs to benefits to Flow versus TypeScript, but we figured out Flow for the most part and solve some of the challenges we had with it.

**[00:41:12] JM**: Do you have a complicated caching system for the frontend also? I'm just trying to imagine the data flow in my head right now and what I’m imagining is you've got a frontend that you have all the benefits of a V8, and the frontend is talking to this backend in-memory data model representation, which I’m sure is its own really interesting application. Then that’s periodically getting snapshotted to Mongo. Is there any significant infrastructure in between that browser representation and that in-memory model on the backend?

**[00:41:48] VM**: Yeah, we do a bunch stuff with Redis to keep things fast. As a site is being designed, you actually don't want to go to Mongo a bunch of times. So there's kind of like this middle layer that's much faster that acts as like an intermediate sort of cache.

**[00:42:03] JM**: Do you have another in-memory model sittings – Or I guess that's the browser, right?

**[00:42:08] VM**: Yeah. In the browser we do kind of to a degree sort of like progressive loading. We don't load the entire app. So if you're not using a certain part of it, only when you go there does it load that bundle, or at least where that bundle doesn't – It's cached locally. If that part of the app doesn't change, the next time there's an update to, let's say, the style editor or whatever, only a smaller portion of the app has to reload. But there's a bunch of things that we still need to do there to make it even faster. Right now we kind of rely on you having to have an almost brand-new machine with a lot of RAM and there's just a lot more optimizations we can make there.

**[00:42:48] JM**: When I was using Webflow, it was really cool. It kind of required a paradigm shift. I was trying to do things in it and I was like, "I don't have an intuition for how this thing works," because it's not exactly like an IDE. I mean, it's kind of like Sketch, but it's not exactly like Sketch. I'm terrible at Sketch also.

**[00:43:09] VM**: That's what I meant by mindset shift. For graphical designers, have to understand like the concepts behind the web. Things like the box model to really grasp what Webflow – How to use Webflow. We almost have to come up with a different name for what Webflow is, like a visual development environment. It's like an IDE, but it's visual. It's almost like what Visual Studio was supposed to be way back in the day in HyperCard and –

**[00:43:30] JM**: I do think it's kind of a category creation.

**[00:43:33] VM**: Yeah, it definitely is.

**[00:43:33] JM**: It's pretty novel.

**[00:43:36] VM**: Yeah. I wouldn't say it's like hyper novel, because there is a lot of like PowerBuilder and all these other tools that sort of like saw this as an obvious step in software creation.

**[00:43:45] JM**: Those were all broken promises, right? That's what they boiled down to.

**[00:43:48] VM**: Yeah, and I think it's sort of unfair to like say that –

**[00:43:52] JM**: Not promises. Broken attempts.

**[00:43:54] VM**: Exactly.

**[00:43:55] JM**: They tried.

**[00:43:55] VM**: I think they were just way too early and the web was growing so fast.

**[00:43:58] JM**: Hey, man! You were too early. You tried it four times.

**[00:44:03] VM**: Yeah, exactly. But even now to some degree, we're kind of too early, because there's just still a lot of like innovation and development. Like developers trying to figure out what is going to stick. What's not going to stick?

So a lot of it is a timing thing. But a lot of it is also because those tools were not able to capture like the kind of go-to-production readiness of what a developer could do by hand, that sort of what gave them bad rep. That they would always say, "Because I created this site in Dreamweaver, or FrontPage, or –" Actually, FrontPage is a bad example, because it's mostly code-driven. But like Weebly and Wix and whatever. There was a sort of like this – Because the code is not clean and it's not as perform as what a developer would do, that gave a lot of like – It eroded a lot of trust that tools like this could work. So our job right now is to keep sort of like pushing the envelope of like, "Hey, you can actually do all these stuff, but you can also do it correctly the way that a developer would do it."

[SPONSOR MESSAGE]

**[00:45:07] JM**: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform so that you can get end-to-end visibility quickly. It integrates seamlessly with AWS so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes.

Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that fuzzy, comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

**[00:46:12] JM**: What's the boundary of possibilities today? What can't you do? What are the things that you wish were possible but are pretty jenky?

**[00:46:19] VM**: So we like to stick to abstractions that are we don't invent any primitives on top of CSS, for example. So Webflow is just like – You can do radial gradients, or background images, or whatever, but the same way that you can do them in CSS. So pretty much everything that you can do in CSS, you can already do on Webflow minus a bunch of like properties that aren't for prime time yet. So there's a bunch of like back clip or CSS shapes. Those things are just not – They don't either don't have browser support yet or just not enough demand that people actually use them a lot in sites.

We used to have a big gaps, like didn't do Flexbox, or we didn't do Grid. So you sort of had to rely on these like old-school kind of float-base layout techniques to rebuild what you could have built in code faster using like more modern layout technology. But now we're not caught up on those.

In fact, I think a lot more people on an absolute basis will be successful with using the power of CSS Grid with tools like Webflow than just natively through CSS. The adoption curve of like Flexbox took a long time. like CSS Grid has also taken a long time because the complex kind of

layout to what Flexbox still needs. I know a lot of developers at Webflow are some of the best developers in the world who have like a cheat sheet for Flexbox open, because the syntax is so hard to remember all the time.

There's like a lot that we don't do on the CMS side, for example. If you had custom code or a custom like code-driven CMS, like WordPress, you already have like all the features that WordPress built over the last 15 years. A simple one I can mention right now is like scheduled publishing. We're not building that until like December. So you have some of these like limitations that we're kind of playing catch-up. But for the vast majority of people who are like Webflow customers, for them it's the difference of like, "If I went to this other kind of codebase driven approach, I'd simply can't do it." It's literally the difference between like not participating in creation for the web and having like a few small limitations that we currently have, but we're like very quickly catching up on.

Oh! The major, major limitations in moving beyond websites are things like in Webflow, you create a website, but you can – For example, right now, do multi-language, because we don't support it yet. So you can't go into without some hacks. I mean, you could do these like JavaScript-driven kind of translation things, but it's going to take us a while to do like true multi-language.

You can't graduate from like full-on website to full-on application where somebody – Let's say you were listing like the most popular – I don't know, products or something like that and all of a sudden you wanted to have an idea to make it more like Product Hunt where you can like login and see things that you uploaded. You can't build that with Webflow right now, because we don't have this abstraction of a user account or authentication. We will in the future, but it's sort of like you're kind of limited to the web design, web publishing space right now for the most part.

But the same core primitives of like how you build software where it's like the domain or the database and the UI and then eventually we'll have like business logic visually. Once you cover all those three, it's game over. You can build almost anything that you can build with software, because the vast majority – Like you said before, the vast majority of things that developers do today, they're like repeating. Almost everyone is building their own like authentication system.

Almost everyone is building their own kind of like database and building their own like way of a building UIs, etc. That can like be really, really streamlined. Where just fewer companies are repeating themselves and they focus on what truly matters, which is like innovation and the problem that they're actually solving for customers, not like innovation in like how they do it in the sort of implementation state.

The way to think about that is just like Amazon AWS. That's a simpler problem, but it essentially abstracted away a lot of skills that people used to be proud of and used to be in almost every company. Every company needed to have like a server rack and a farm and like skills around like how do we order and replace hard drives and how do we know they're failing. How do we configure rate? How do we make sure that there's power redundancy and all these stuff. You needed specialized skills that every single startup in every single company to do that. Now, all that complexity is behind an API. Essential like, "I want compute power. I want like a place to store stuff." Then Amazon, or Google Cloud, or Azure, or whatever, they take care of that stuff.

I think we'll get to similar sorts of primitives for building like richer parts of the software development stack. So you don't have to worry about where do I write the code for these React components? How had I deliver them to NPM, or how I deliver them to like this production environment through continuous integration, continuous delivery, or whatever?" You have designers, like have a living and versionable design systems that developers can pull in and it's sort of like you have like a hosting infrastructure sort of automatically figured out for that entire piece.

So then software development can become more like – I don't know if you've ever used Shopify. Everyone used to build their e-commerce sites from scratch. You controlled your like payment gateway, etc., etc. Now, Shopify is essentially like e-commerce as a service. They take care of hosting. They take care of the payments infrastructure. They take care of a lot of compliant stuff. You essentially bring your products and your design, and like their design piece still requires a lot of programming. You have to know how to learn Liquid, and they have a lot of coding requirements, which is why when we built our own e-commerce engine it was basically Shopify plus our visual development tools so you don't need that piece.

 

But Webflow can become just like Shopify as for like the infrastructure for running all the things you don't really want to worry about. You just want to worry about your products and your customer experience. Then you just manage that. They take care of everything else. They take care of scale. They take care of like you have a big sale. You don't have to worry about like your server is going down. It's Shopify worrying about like running this at scale.

We believe Webflow can get to that level for like all kinds web applications, where it's like Heroku on steroids. Where you're essentially like declaring what your software looks and functions like and what its data model is, and then you're basically building the things that are very specific to your business, and that's usually things like how you actually solve the problem. What the application looks and feels and functions like rather than what servers does it run on? What's the frontend framework? What's the backend framework? Etc. You're actually worried more about the true guts, the true like business like value-added that your product or service brings. That becomes a lot more important, because you sort of don't worry about the infrastructure.

**[00:53:02] JM**: I love it. Why did this problem obsess you?

**[00:53:07] VM**: Two things. I saw so much frustration with designers having like amazing visions for product or service and just not being able to make them real. One of them is my brothers, who's my cofounder, who created a product that's very similar to GoFundMe and Kickstarter. It was called Help Riot, where he wanted to help – Essentially create these like social campaigns, fundraising campaigns around social causes that were underfunded.

He had the whole thing designed. It was amazing, and he just couldn't find a developer. So it died. It like literally died. So that idea, that passion, that everything like died with that with his inability to not find a developer or learn how to code. That's actually the energy we took into creating Webflow. How can we get Sergie, my brother, to be able to create that in the future?

**[00:53:57] JM**: What a perfect model application to be thinking about for all those years that you were beating your head against the wall, like trying to make this work, right? Because it has a lot of simple things that we can now say as simple nouns, crowd funding, or just payments, or accounts. Yeah, like you said, these simple primitives. Everybody in the United States knows

how these things work, knows what they are. Yeah, you can create an account. You can remove an account. We all grew up with the Internet. Why can't we instantiate these things with the click of a button?

**[00:54:31] VM**: Boom! Exactly. Exactly. There's no reason why that shouldn't be possible. My daughter right now, her school, her public school, has this thing. It's called focused learning goals. That's the things where they really want to like learn and grow in a specific area, and the teacher helps them kind of move along those goals. Her goal is to keep improving her. She used to keep it on a journal in her paper, like weird animal facts. It's essentially like her internal social network that she created with all of her friends. She collects these facts from her friends.

Last year, like six months ago –

**[00:55:05] JM**: There are a lot of weird animal facts out there.

**[00:55:06] VM**: Yeah, exactly. It was sort of like how does she get that out into the world? Instead of just sharing like her journal with just the friends that she has. Six months ago, one of our dates, we have these recurring dates. She asked like, "Hey, dad can you help me create a website for keeping track of these?" I taught her these abstractions of like, "Hey, an animal is an object. An animal has a – A fact is an object, and an animal can have many facts." So all of a sudden –

**[00:55:33] JM**: How old is your daughter?

**[00:55:34] VM**: She is 10 right now. She was nine at the time. So I'm not teaching SQL. I'm not teaching her Mongo. I'm not teaching her kind of like how to install a database server. I'm teaching her concepts that are very human and like easy-to-understand. She understands that relationship. She understands essentially like schema modeling from just the relationship between an animal and facts. Then she's like, "What kind design do you want?" She sketched it out, then we sort like started to build it and kind of explaining these concepts of like, "You can't just like drag a box in. You have to think about how this is going to appear. If you have two animal facts stacked side side-by-side with each other on larger screen, you have to start thinking about what's going to happen when you see it on a phone. Kind of like there's not

enough room. So you have to get one to push the other one down." How to explain those rules to the Webflow, in this case.

Now she's super excited to like share this website with all of her friends and like – But she's still limited, because her friends have to bring facts to her and she has to write them down, and like at home she has to go login and go enter them. What if her friends had a form on the website to say like submit facts that she can then create? She already like explained this process to me. I want my friends to send me facts and I want like to approve them, and I click when I approve them if I think they're funny. Then they go live on the website.

That's already an application, right? Because then you have to like login and authenticate. Your friends have to create accounts.

**[00:57:00] JM**: Oh! So you can't do that yet?

**[00:57:01] VM**: You can't do that yet. But, already, you have like this application of something's like kind of like silly and simple, but already it's like the power software and the power of the Internet to like the fact that there are so many of these silly applications means that there is probably a bunch of non-silly things in the middle that are just like waiting to be built.

One analogy I sometimes give is where we kind of are in the world today, like Steve Jobs gave this example many, many years ago, of hundred years ago, the telegraph was the main way that we shared information. You had to go to a telegraph operator and like say, "Hey, here's the message I want to send my friend," or whatever, then it was sort of like translates your Morse code. It goes over the telegraph wire. The operator on the other side sort of like, "Here's the Morse code." Translates it to paper. Hands it over to the other person, which is fine, right? Those are first step into communication across long distances.

But then there's a bunch of businesses that were formed around like the commercialization of the telegraph, and the thought process was, "Well, hey. It's annoying to go to the post office to sort of like rely on them being open. Why don't we bring the telegraph into the home and give people like a manual on how to learn Morse code? That way we remove some of the barrier of like having to walk like through the rain or whatever. But there was still this like – It was missing

the forest from the trees around, “Okay, now we’re going to teach everybody to write Morse code, or to like communicate with Morse code, like tap, tap, dot, whatever.” That's where we are right now with software development. People are saying like, “Oh, the way to participate, the true way to participate is to learn how to code.” That’s why we have like code.org and we’re trying to teach code everywhere.

I think that’s a wrong layer of abstraction. It's sort of like saying, “In order to get into computing, you have to learn zeros and ones to learn how registers work and learn how like the assembly languages optimize things for certain architectures.” I think it’s totally the wrong abstraction layer.

The correct abstraction layer to how human communication was solved was not the telegraph, not teaching people how to write Morse code, but to solve that same analogous problem with something that's a lot more human compatible, like Alexander Bell and others then created the telephone, which is essentially using an interface people already had and were very comfortable with, which was their voice. You could automatically translate your voice into digital signal or like analog signal at the time and have somebody pick up the phone on the other side and have a conversation.

Same actual underlying problem solved in a much more scalable way that’s available not just to people willing – That already Morse code or willing to learn it or able to learn it and have access to the ability to learn it, which to learn code is like orders of magnitude harder than learning Morse code, right? Because it’s not just like 26 letters and like speed and practice. It’s like frameworks, and concepts, etc. That's where we are right now. We have to get people to believe that, yes, it's important to build more software, because it's so valuable. Software is eating the world. Everything we’re doing right now is driven by software. The program you're probably using, the way we schedule this, the way the websites we visit, etc.

**[01:00:01] JM**: The way I ordered this coffee.

**[01:00:03] VM**: There you go. Exactly. That's sort of like the shift I see no code taking us to, to say, “How do we get more than 20 million people building software? How do we get a billion people?

**[01:00:12] JM**: I agree with that. That's why I have “fed into the hype”, as I think some listener told me.

**[01:00:18] VM**: I don't think it's hype at all.

**[01:00:19] JM**: I don’t think so either.

**[01:00:20] VM**: I don’t think it’s hype at all. I think people who are saying it’s hype were the similar category of people who are saying the Internet was hype.

**[01:00:28] JM**: Totally. Okay. So you got to go soon, because you’re building – You’re bridging – We need you to get back to bridging the gap between the hype and reality.

**[01:00:37] VM**: Yeah.

**[01:00:40] JM**: So one of the things I found pretty interesting, I listened to couple interviews that you've done, and one on Design Details and the other on This Week In Startups, which I like both those shows. This Week In Startups in particular has been really influential for me. Jason's interview with you was awesome.

There's a couple things you said in those interviews where you kind of alluded to the fact that you couldn't really have built Webflow if you hadn't gone through this – I don’t know if it was like nine months, of there’s some period where you were just working relentlessly and you got the kind of flow state. I’m familiar with this. I mean, the early days of Software Engineering Daily, I wasn’t building Webflow, but like was just only working, only working, basically and I loved it. I couldn't have been happier. But it was – I looked back and I imagine how those working relative to my personal life and like loved ones and stuff. It’s just devastation. Just like –

**[01:01:35] VM**: Yeah. There’s no way – it's unsustainable. You can only do it for a season. We had to do it out of desperation. You literally didn't have money and income, and it was like this is just the sheer amount of work that need to be done and the time available to do it or the resources, like the people available to do it. It was just two of us, or three of us at some point.

So there was just no other choice. But that was our choice individually and we had to get like our partners and our families. I had two kids at the time. We had to get our families to buy into that.

**[01:02:04] JM**: You did that upfront?

**[01:02:05] VM**: And know that it's a season. Well, actually, did that upfront for like three months and they had to keep extending. But I personally couldn't do it for more than a year. It was just getting very, very close to burnout, if not like true burnout.

The key difference, a lot of times you hear this sort of like hustle culture, where people expect everyone at their company to work that way and sort of like devote their lives. I think that's a totally different kind of thing. When you're individually working on something that you're super passionate about and you're the one buying into that, that's a totally different thing than the expectation from an employer to work that way. I think employers who expect their teams to work that way in a way that is just like very like singularly focused, you're not focusing on your health, you're not spending any time with your family. They're going to build foundationally like lasting teams. They're not going to build like cultures that can like innovate and inspire others to join.

**[01:02:58] JM**: What's funny, I worked at Amazon for eight months. Amazon gets this out of people, but incidentally. So Amazon paints a picture of like a brilliant, fantastic future, and it's inspiring enough that some people actually work that hard sometimes to their detriment, sometimes not.

**[01:03:14] VM**: Yeah, I could work. It could work. To me, personally, it just wouldn't feel good.

**[01:03:18] JM**: No. All I'm saying is they don't – Those perceptions that you read about at like that New York Times article about like the –

**[01:03:25] VM**: They're cripple at their desk or whatever.

**[01:03:27] JM**: It's not like that. I mean, it's weird, because the culture develops where you see a lot of people doing that and some people get the impression that like that is how they have to work. It's actually not how you have to work at Amazon. Amazon's totally realistic about the fact that, yeah, these people aren't capturing 99.9% of the upside. You and your [inaudible 01:03:48] raised some money at that point.

Anyway, it's 11. I think you got to go. But I found those admissions to be honest. Yeah, I mean I think like that flow state is satisfying.

**[01:03:59] VM**: Yeah, it is definitely. So that's one of the things I miss about coding, just like being in that kind of building mode.

**[01:04:06] JM**: Inbox doesn't do it for you?

**[01:04:08] VM**: Nope. Not at all. I mean, there're different layers of satisfaction now. When you empower your team to build a lot more things in parallel and you're there to like support them, to hire more people, to like augment, like help them be successful. That's a whole different level of satisfaction, because you get to see what they build collectively, which is way more than I could personally build getting into even 24 hours of flow state.

Even if I could do that, it's just physically impossible to innovate that much as much as a team that's well empowered, has autonomy, has like a level of ownership where they understand kind of what they're building towards and have the sense of purpose of like what their building is important. You get way, way more. You make a bigger dent in the universe than like what you can do individually. That's kind of the definition of teams, right? The reason we form teams is that we can do more together. Not because like we want other people like carry our water or something like that to make our burden easier.

Sometimes that's the case. When you're literally working like crazy, it's good to like have another person come in for reinforcements. But the reason a team is formed is because we like to draw on the strengths of others that we don't have, and together you end up building something that's much greater than you can build just by yourself. For me, that tradeoff has been like really satisfying even though I don't get to code pretty much at all these days.

**[01:05:33] JM**: Vlad, thanks for coming on the show. It's been awesome.

**[01:05:35] VM**: Of course. I really enjoyed it.

[END OF INTERVIEW]

**[01:05:47] JM**: Monday.com is a team management platform that brings all of your work, external tools and communications into one place making cross-team collaboration easy. You can try Monday.com and get a 14-day trial by going to Monday.com/sedaily. If you decide to become a customer, you will get 10% off by coupon code SEDAILY.

What I love most about Monday.com is how fast it is. Many project management tools are hard to use because they take so long to respond, and when you're engaging with project management and communication software, you need it to be fast. You need it to be responsive and you need the UI to be intuitive.

Monday.com has a modern interface that's beautiful to look at. There are lots of ways to use Monday, but it doesn't feel overly opinionated. It's flexible. It can adapt to whatever application you need, dashboards, communication, Kanban boards, issue tracking.

If you're ready to change the way that you work online, give Monday.com a try by going to Monday.com/sedaily and get a free 14-day trial, and you will also get 10% off if you use the discount code SEDAILY.

Monday.com received a Webby award for productivity app of the year, and that's because many teams have used Monday.com to become productive. Companies like WeWork, and Philips and Wix.com. Try out Monday.com today by going to Monday.com/sedaily.

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

[END]