EPISODE 851

[INTRODUCTION]

[00:00:00] JM: Software engineering is an art and a science. Manage engineers is to manage artists and scientists. Software companies build practical tools like payment systems and messaging products and search engines. Software tools are the underpinnings of our modern lives. So you might expect this core infrastructure which modern humans rely on to have been constructed with pure formulaic rigor.

But the best software tools are not built within a totally defined process. Software is built through messy iteration. When a piece of software looks pristine, that is often a function of how many mistakes have been made along the way and subsequently been corrected for.

There is no fixed process for how to build good software. As our tools get better, we have to update our software engineering practices to utilize those new tools. We have to rethink the style that we're working in. We have to discard old tools and procedures in order to pick up the new ones and have higher leverage. As an organization scales, the structure of the organization needs to be modified. Team members need to be reallocated. Checks and balances need to be put in place. Rules and cultural practices need to be codified. Because a larger organization cannot have ties broken by an individual, we need these sets of rules to manage the increase in scale.

Software is built by humans, and every management decision must be considered in the light of human psychology. When we change a line of code, the code does not get emotional about being altered, but the same cannot be said of humans. Even a minor conversation between an engineering manager and a direct report can have lasting implications on the relationship between those two individuals.

Will Larson is the author of *An Elegant Puzzle: Systems of Engineering Management.* Will works on foundation engineering at Stripe and he's worked in engineering management at Uber, Dig and other software companies. Elegant Puzzle provides strategies, tactics and ruminations

on software development. Will joins the show to explore the multi-faceted subject of engineering management.

Before we get to today's show, a few updates from Software Engineering Daily land. FindCollabs is a place to find collaborators and build projects. If you're looking for a cofounder or other programmers to build your project or your company with, check out FindCollabs. I've been interviewing some people from the FindCollabs community on the FindCollabs Podcast. So if you want to learn more about the community, you can hear about it on the podcast, which is linked to in the show notes.

We also have a new app for Software Daily on iOS. Android is coming soon. The app contains all 1,000 of our old episodes. You can find topics that you're interested in, whether it's Kubernetes, or Bitcoin, or shows about Stripe, and you can comment on episodes. You can have discussions with other members of the community. You could become a paid subscriber and get ad-free episodes. Just go to softwareengineeringdaily.com/subscribe if you're interested in subscribing.

Now, let's get on to today's show.

[SPONSOR MESSAGE]

[00:03:43] JM: DigitalOcean is a simple, developer friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow. DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support, plus they've got 2,000+ tutorials to help you stay up-to-date with the latest open source software and languages and frameworks. You can get started on DigitalOcean for free at do.co/sedaily.

One thing that makes DigitalOcean special is they're really interested in long-term developer productivity, and I remember one particular example of this when I found a tutorial in

© 2019 Software Engineering Daily

DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence, and an attention to just getting developers off the ground faster, and they've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's the D-O.C-O/sedaily. You will get started for free with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:05:44] JM: Will Larson, welcome to Software Engineering Daily.

[00:05:46] WL: Thank you so much. Glad to be here.

[00:05:48] JM: Why do engineers need managers?

[00:05:50] WL: That's a great question, and I think in a lot of smaller companies, you don't, but at some point you get to have so much complexity, so much change that needs to be kind of managed through, that if you don't have managers, then people spend their entire time managing through that change, that complexity themselves, and they really aren't able to focus on kind of the work they want to be doing, the work that's maybe most valuable for the company. So, really, people to help navigate change is why you need managers once you get to a certain scale.

[00:06:17] JM: In Google's early days, they tried an experiment where they removed all the managers. Larry Page said, "Let's get rid of all the mangers." It didn't work out so well for Google back then. Today, there are much better tools for engineers to self-assemble. Do you think Google could work without managers today?

[00:06:39] WL: One of the kind of special perks of working at Stripe is actually the individual who was the manager for all of engineering at Google. During that period of no managers comes in and tacks with us occasionally. So, we've gotten to – I can't share his insights,

Transcript

because those are his insights to share. But I think maybe it was a difficult experiment for everyone involved at that early stage.

Stripe actually for a long time as well really had very few managers. We, for very long time, until about three years ago had almost no eng managers, just two or three for a very large team. So we've definitely kind of tried that experiment and learn from that experiment here as well. I really think that it just depends on your company. I think one of the most exciting things that has changed in kind of Silicon Valley last decade is just these number of kind of hyper scaling companies and these companies just so much changed. They're going to new countries. They're on to new products. The teams are growing 2X, 3X, like 4X year over year, and you really do need people involved to help manage that type of growth and help people find how to adapt to that change.

I think tools are great, but one of the problems with tools is you like create them and kind of set them up for a certain circumstance. But I think what's novel about managing today in these fast growing companies is that these circumstances are changing so frequently that the tools are out of date all the time.

I think a joke I've heard at my current company and also my previous one is that the tools and kind of systems were at least six months behind all the time. I think that is true when you're growing this quickly. The tools just take a while to catch up, and that's where having humans who kind of are specialized and thinking about this stuff is so valuable.

[00:08:18] JM: So you have the tools that are out of date constantly. What about the engineering practices? Do those constantly get out of date also?

[00:08:28] WL: I think for engineering practices, if you're thoughtful, you often layer more on top versus like replacing or throwing away. For example, if you have two people, you can sort of do whatever you want. You maybe have like a paper to-do-list. Once you have 10 people, you probably want like a sprint planning come on board. You want some sort of way to coordinate what you're working on.

As you grow more or maybe you want like quarterly or annual planning or something, but you don't throw away the sprint planning. You still need the spring planning. Maybe the individual still have their to-do-lists at the very bottom. So it's more about layering on the minimum process you need to be effective versus like throwing away and kind of starting over from scratch each time.

[00:09:04] JM: But under that rubric, how would you know if you should throw away spring planning?

[00:09:10] WL: This is a really interesting question. So, I think there's the idea – So a lot of management is actually like governance, like actually governing kind of like a group of people. So, one of the I think ideas in governance that you don't see often is this idea that every law you pass should actually like spire in like three or four years, or in 10 years or something. What would it look like if taxes expired after 10 years and actually had to get passed again? We'd have a much cleaner book of laws. I don't know if the laws would work that well.

I think when you come back to the processes that we used for our own teams, we should probably think about this in a similar way, or we should on an annual, every two years like really reflect which of these services are – Which of the processes are still serving us well. But it can be hard to figure out which, because I think one of the biggest kind of sins of management is unnecessary change.

Often, when people come into a new role, the first thing they do is they change something. It's such a strong pressure to show value that you wanted do something useful. I think it's really important not to kind of jump in and change things until you understand. I think there is like a one hand, like a strong value to getting rid of things that don't work, but I think so often we just change things to try something different without a structured experiment, without a lot of clarity to our purpose. I really think to your original question, we should be reevaluating, but we should be clear about our goals, why we adapted it and understanding like, honestly, the friction that caused us to consider getting rid of the current process is super important not doing it in a haphazard way.

SED 851

Transcript

[00:10:45] JM: One problem I see here is like if you have a business where things are going really well, like Stripe, you are going to be tempted to, in terms of management strategy, go through this basically loss aversion where you say, "Okay, look. Things are working really, really well. We've got sprint planning. Nobody like sprint planning. We've got the quarterly or we've got the weekly analytics meeting. Nobody's paying attention in the weekly analytics meeting, but we've seen it add value in the past. Let's not break what's working." You can get into this state of loss aversion.

What about an alternative model where you start to say, "Okay, look. Every now and then, we're going to say, "Okay, one out of ten teams at stripe, you have to drop one of your engineering religious practices," and then you have to like document what happens when you drop that religious – Maybe I'm just playing like an armchair quarterback, because I've never managed any team or something the size of Stripe. But in hearing what you're saying, I'm hearing like, basically, if it ain't broke, don't fix it.

[00:12:07] WL: So, I think there's a certain size you get to and can actually run like controlled experiments and actually learn from them. I think there is like a storied history of using "experiments" to prove your point that don't actually have a lot of rigor behind them, and a lot of experiments you can actually see what the conclusion will be before you even start, because it's how the experiment is structured. It's like, "Oh, this team that's deeply like excited about Kanban is going to try Kanban." They're like, "Hey, this team is really bought in to Kanban. They're going to be successful with Kanban," more successful than they're using Scrum or something which they hated.

So, a lot of times I find that the attempts to bring rigor to the sort of decision-making can be kind of a masquerade around kind of just like a personal preference. I think it's really important not to let that sort of thinking I come in.

On one of my favorite books in the last few years is *Accelerate*, and it really brings a lot of science of actually measuring how teams perform with different practices. This is from the kind of the same kind of spiritual vein as like Slack and people where these are great books that actually bring data and [inaudible 00:13:16] science and like experiments that are controlled to actual best practices. Definitely, I'd love to see more of those.

SED 851

Transcript

So often, I think it can just be a bit of a crapshoot when people change processes. That said, being like really thoughtful, rigorous, something I do every quarter is a look on my calendar and I throw out meetings that aren't working for me anymore. We should also be doing the same thing for our practices organizationally each time we grow an order of magnitude. A lot of our planning, a lot of our headcount, a lot of our kind of like structural things won't work anymore. Let's make sure that we really want to bring those with us as we move to the next order of magnitude and kind of engineering headcount, not just like bring them with us automatically. I think that's a great idea.

[00:13:56] JM: You worked at Uber during its rapid growth from 200 to 2,000 people over two years. How did you manage your own psychology during that time?

[00:14:08] WL: I think the best part of rapid growth is like pretty much every week you like raise your head, you look around and there's like some like really smart, talented person who is sitting next to you who wasn't there the week before and can help. So, during change, I think you have to really make sure that you stay open. Once you start wanting to control the change too much, you start trying to shape it and then it will kind of overcome you. But if you look at how to direct it a little bit, how do you help these new folks who are joining be successful? How do you look at facilitating change versus like controlling change I think is really powerful?

That said, I think there's just like basics that people know to do but don't to. Just like actually taking vacation, actually cutting off at like 6PM or 5PM or whatever as like working through your schedule. I think early on in your career it can feel very selfish to take what you need to continue to remain present and happy. But I think if you aren't selfish in that way, you get to a place where you're like quite unhappy. Those around you can tell you're unhappy and you actually like kind of damage your career in a given company.

So, I really think it's thoughtful to understand what you personally need and taking it so that you can actually be the person you want to be for longer versus just like peeking like the first three months of your new job or something like that.

Transcript

[00:15:29] JM: What were the chronic mistakes that you made as an engineering manager in year earlier stages of career?

[00:15:37] WL: The biggest mistake that I've made just really over and over is not being able to distinguish between something that is a one off and something that we're going to keep working on and evolving for like years. For example, let's say that you really want to get a project staffed. I think there's a way to go into like the planning process and kind of feel like either this project get staffed or it's a disaster for me personally, like I care about this project so much.

But another way to look at it is like, "Hey, I'm going to try to get this staffed every quarter for the next like two years and keep having this conversation." Instead of looking at things as this like kind of one-time kind of event, but instead looking at them as this evolution of like a relationship and of this like ongoing like attempts to accomplish something. I think one way to look at it creates conflicts. You have to get it now. You're fighting to get this outcome.

The other way is like, "Hey, together we're going to evolve our understanding of this over time," and it takes all this pressure off of it because it's not like you do it now or you're ruined or doomed or something. Instead it's like, "Hey, I want it to happen this time, but if not, we'll have the conversation again," and it just takes this pressure off it where I used to just feel every decision was this really essential, critical thing that had to be perfect. In retrospect, I was just like consistently wrong that every single one of those critical decisions that had to be made just once. All of them have been able to like reopen, keep thinking about and learn more and remake it overtime.

[00:16:58] JM: Software engineering is a very immature field. We just certainly have not had this field around for very long relative to how much impact it's having on the world and how many people are doing software engineering these days. So we have this kind a way of developing engineering cultures where we oftentimes just kind of like a look at, "Okay, what's the company that has been successful the most recently? Let's just copy the stuff that they're doing in terms of engineering management, reengineering practices, because we've got a product that's working. Let's just like figure out like the basic engineering management framework that will get our product to continue and not fall in space."

That pattern of kind of just replicating the previous success story, that can lead to – I mean, it is a great way of protecting downside risk, but it can lead to the propagation of like anti-patterns or memes that shouldn't be copied. Are there any memes in the engineering landscape that you see relentlessly propagated that you disagree with?

[00:18:13] WL: I think this idea of cargo culting ideas without understanding kind of the environment that they came in is a very important and very, very prevalent. In fact, I think going back to the previous idea kind of new leaders trying to show value early in their careers or early in their start of a new job, I think, typically, cargo culting idea that works really well previously for them is kind of the most common failure case for them. Where there's like, "Hey, they don't understand all these auxiliary pieces that made it succeed at their previous company."

In terms of like pieces that I think are kind of particularly important to think about that maybe you don't think about enough. One that's really been on my mind a lot recently is you have this kind of like dual path for engineers, you know, to management and stay senior as an IC. Now, everyone knows to say that, but how do we make sure that it's really true from a disparate distribution perspective, from a compensation perspective, from a career pathing perspective that both of these paths work out really well for folks?

I think as an industry, there's still a lot more work for us to do there for kind of staff and staff plus level engineers to fill the level of kind of impact and contribution that kind of managers often feel. That's definitely one.

I think the whole kind of – The entire roles of engineering manager, technical program manager, product manager, where those boundaries are. I think this is something that we kind of like port from place to place, but is actually very different each place. So I think that's like one place where companies kind of feel like they actually have the same thing kind of ported over, but actually in practice have radically different implementations of it.

Although we kind of talk about like a product manager is a product manager is a product manager. In reality, the roles are radically different and the same is true for engineering management where it's certainly – One of things I love about Stripe is we have a very broad church of product management where folks get to do like certain engineering management

rather and product management too where were folks get to do just such a broad view of things. Wherein sometimes these roles almost feel like general management roles. They're really kind of running like real product lines. That's something I love about Stripe, but certainly like sometimes when people like initially start, they're just surprised if it's your breadth of the role that are coming into them.

[SPONSOR MESSAGE]

[00:20:39] JM: Cox Automotive is the technology company behind Kelly Blue Book, autotrader.com and many other car sales and information platforms. Cox Automotive transforms the way that the world buys, sells and owns cars. They have the data and the user base to understand the future of car purchasing and ownership.

Cox automotive is looking for software engineers, data engineers, Scrum masters and a variety of other positions to help push the technology forward. If you want to innovate in the world of car buying, selling and ownership, check out cox autotech.com. That's C-O-X-A-U-T-O-T-E-C-H.com to find out more about career opportunities and what it's like working at Cox Automotive.

Cox Automotive isn't a car company. They're a technology company that's transforming the automotive industry.

Thanks to Cox Automotive, and if you want to support the show and check out the job opportunities at Cox Automotive, go to coxautotech.com.

[INTERVIEW CONTINUED]

[00:22:00] JM: The path for scaling yourself within a company, you're talking about different pathways for an engineer. If I understand correctly, you're talking about the path to becoming kind of a principal engineer or like a staff engineer or like just kind of this Jeff Dean kind of person that's like a really, really good programmer and builds really, really useful internal tools versus becoming like a VP manager or a senior manager ascending that hierarchy.

SED 851

Transcript

There's also a third category of scaling yourself within a company as an engineer, which is you write a six pager or you use your 20% time to build some innovative product. Those categories of engineer can really move the needle on a company by standing up an entirely new business line. Is there a way to encourage people to go down that path? I mean, obviously, you can say the 20% time, you can say we've got this six pager process. You can also do more. You can push people harder. You could say, "We have 50% time." You could say, "We have 80% time." Only 20% of your time should be business logic. 80% of your time should be innovating on cool crazy stuff. How do you foster that kind of innovation?

[00:23:15] WL: I have like lots of kind of different ideas around on this one. I think one thing I think about is that any business has a lot of like just forced work in it. So, that can be scaling work. That can be maintenance work. That could be work like migrating to meet the full GDPR requirements or the recent kind of California privacy requirements that are coming into play. These are things that just have to happen.

So, vendors also work that's kind of discretionary work that's actually often like more fun where you're like, "Should I start a new business line or should I like port GDPR compliance software?" Well, probably, I really just want to start this new project that I bring here. So I think one of the challenges is like businesses have this large amount of like forced work that you need to make sure you value people doing, because the business just can't function without it.

Then the other hand, there is this kind of like unknown wildcard quantity of like this potential to kind of do experiments or explorations that turn out to be like radically valuable. So I think one of the challenges here like how do you create the right incentive structures for kind of encouraging this. Another aspect that I think is like easy to miss sometimes is that different people kind of by nature of their backgrounds, by nature of kind of their experience in the industry or kind of their financial safety net can afford to take risks like that where they're going to do an experiment that's maybe huge to the company and maybe goes poorly. Some people feel very safe doing that and others don't. So I think you can inadvertently kind of perpetuate kind of class systems in your company where people who are enabled to do that have this one experience and people who don't feel safe for whatever reason, and there are many legitimate reasons not to feel safe taking this sort of risk professionally, kind of end up doing this less exciting, less recognized toil work.

SED 851

Transcript

So, to me, that's really a question of how do we make sure that we have the incentive systems right where we can actually enable kind of everyone to participate in an equitable way around these sorts of experiments. One thing that we've been trying out here and a bunch of other companies have is the idea of like rotations where every one, two years, you could do a three-month rotation on to another project and actually like kind of pick whatever you might want to work on and kind of do an exploration, learn a bunch there and then coming back to kind of the team event, like learning, growing, supporting more rigorously.

Just like what one other quick corollary here, and similar to this is like mentorship, where like what is one of the ways we identify senior engineers is how they help others grow around them. So, making sure that we kind of recognize that work, which not necessarily easy to be mentoring a huge team and to be kind of off on the side, like kind of doing this kind of innovative thing. So, really, one of the challenges as I see it is there's all these different behaviors you want to recognize and reward on and how do we make sure that we balance the different paths in an equitable way just gets to be quite complicated. So, to me, it's just designing incentives is incredibly difficult and complicated, and if we can just figure out that, then getting this model where we have more people doing innovative work more of the time and reap the rewards of that is much more tenable.

[00:26:24] JM: But the problem – What you're describing there with wanting to have an equitable distribution of work between boring legacy services that have a lot of maintenance and GDPR compliance to be written versus sexy greenfield project that might be turning into a new business line is – Whereas if you took an engineer who is working on the boring service X and they're okay working on boring service X, because they've got student loans to pay off or whatever and they feel intimidated. They don't feel like they can go and do sexy greenfield – They don't feel like they can do sexy greenfield service of their own accord, right? But if you, Will Larson, drop in and say, "Hey, look, look. For the next month, we want to go work on sexy greenfield service." They're going to be like, "That sounds great. I'll go do that." But if you do the converse and you say to a person who has decided to go work on sexy greenfield service, because they have some money saved up. They don't have any student loans. They've decided to do that of their own accord and you say to them, "Hey, it's time for you to do your rotation on the boring service." They're going to say, "Okay. I quit. I'm not interested. Why would I do that?"

How can you create an equitable distribution when there are so many boring services to work on and the people who have no fear of downside risk will not do the boring maintenance work?

[00:27:59] WL: I think of two different thoughts here. The first one is that the people I've worked with who I cherish and who are the most technical, most senior engineers I've worked with are folks who have like a strong sense of service and to have a strong sense of kind of duty to creating an environment where folks can succeed and everyone can succeed.

So, it hasn't been my experience that there are these people who are like extremely strong and only want to do the best work and want to horde accord it. My experience has been the best people I've ever worked with are those like, "Here's this amazing work and want to make sure that there is like equitable and like access to it," because it's their values and their values which will make them like great engineers around training, learning, sharing, bringing people up so the organization is stronger around them and not just what they can bring, but what can they help others bring in and grow into bringing to kind of company and the organization. So, I've just like not found the case that people who are out to get theirs are the best folks I've worked with. My experience has really kind of gone the other direction.

[00:29:08] JM: Your book is *Elegant Puzzle*. Why is the book medium a useful format for disseminating information? Why not just do blog posts and tweets?

[00:29:19] WL: So, the first advantage of writing a book is that you can actually touch it, which is like quite – I don't know – One of my coworkers does a lot of woodworking and I for longtime didn't understand why he did woodworking. But printing out these copies of this book and being able to kind of like interact with it physically, there's just something like very special about being able to interact physically with something you've created when you've been working on like a digital element or adigital medium for so long. So, I think there's something really special there just in general.

But in terms of why a book, why not blog posts? I think blog posts have great distribution. Blog posts, you can do kind of what you want. A couple hours on a weekends, a couple of hours a month. You can actually create a blog. For both blogs and books, marketing is actually I think more important than people realize. Getting distribution is actually quite important. Just writing is

Transcript

never enough, particularly if you right for like a more of a niche audience, which is I think mostly true for ensuring management. It's still not the largest audience in the world, although growing all the time.

I think the thing that's special about a book is you can actually have some core themes that layer throughout the entirety of it. So, for my book, it's about engineering management, but really there's like two themes that I think are like really important that layer through like every piece. The first is that we can actually make good decisions, and good decisions matter. I think there's often a sense that management is about process or about kind of on these fundamentals, like great one on ones, career development. These things are like critically important. But I really think that the beauty and the art of engineering management is often about making these really critical decisions about how to kind of make tradeoffs between two teams that don't agree initially. How can you find the path forward where they both get engaged, or when you have constrained resources with headcount or like prioritization, like how do you actually make the right decisions that kind of have impact on like business and actually get the results you want. I think that is something that is kind of surprising under discussed as management. But, actually, quality decision-making is just so core.

The other bit though is that I think if we're intentional, it's actually possible for management to be like this deeply ethical profession where we are making a world for others that they live in that they find rewarding and that recognizes people who behave in kind of the ways that we really want them to and is consistent with our beliefs of like what the world should operate like.

So, I really think getting to explore this in detail, there is only so much you can do in a blog post in terms of just like pushing these ideas in different shapes and different like sizes. I think the book has just been like really powerful for being able to like have a bunch of repetition and different angles on these couple of core ideas

[00:32:08] JM: In your book, near the end you have references to several books, mostly books about engineering and people management. You seem like you've read broadly. What is a book that is not explicitly about management or explicitly about engineering that has had an impact on how you think about systems of engineering management?

SED 851

Transcript

[00:32:36] WL: Going back to kind of the mistakes that I make a lot of my career that we talked about earlier, I think one book that I've read is *Infinite and Finite Games*, which is a great book in the sense the title gives you like 90% of the content and then like there's a two sentence version that gives you 100% of the content, which is that there are finite games that have ends and your goal is to win. There are infinite games that don't end and the goal is for everyone to keep playing. Kind of moving from this worldview where you're trying to win, this worldview where everyone's trying to keep playing together and kind of collaborating, cooperating, succeeding over many rounds as kind of the goal. So, I think that book has really helped articulate something I've been slowly learning over my entire profession, but also my entire, I suppose, life.

[00:33:19] JM: When you were in the early days of Uber and the company was fighting to maintain an increased market share in the well-defined ridesharing market, did that feel at the finite game?

[00:33:34] WL: I think that that's really interesting kind of question to think about. Maybe I think the challenge was at times we framed it as a finite game, like get to market share and kind of succeed, like win market by market. But in reality, I think what you see over time is that it's actually not about winning a market. It's about how do you manage very carefully, how do you cultivate and nurture the supply and demand in each market over time? How do you make sure that the partners are getting paid in a way where they continue to like drive on the service? How do you make sure that the safety and reliability of the service is high-enough so that riders keep using it over time?

I think it's always the case in my experience. They can take something that looks like a one-off event and kind of stepped back and there's actually a much deeper system underneath it that you could be thinking about, and thinking about it will force you to be like much more thoughtful and clear about your thinking versus just thinking, "Go in, get this done, move on." I've never seen any business where you can just rest on your laurels and kind of end with it. It's always you have to figure out how do you maintain what you've accomplished at any given point.

[00:34:44] JM: As a business, when should senior leadership frame things as a competitive finite game? Because we see historical examples when finitude was useful, whether we're

© 2019 Software Engineering Daily

talking about Uber, framing things in the short term as being a finite game and going after competitors, or Facebook framing things in the short term going after competitors. The Google+ example is great. There's that classic story of Mark Zuckerberg standing up in front of the Facebookers and saying, "Look, Google is very clearly gunning for us with Google+. We are going to ship a feature every day or every week until we beat this thing," and he marshaled the troops. That was almost undeniably a useful framing of the world as a finite game.

But more generally, Facebook is successful, because it frames things internally to the culture as an infinite game. So when do you want to use the instrument of a finite game as an engineering manager, engineering leader?

[00:36:04] WL: I think this is something that really becomes a question of a personal style and kind of company and culture. I think there are – To your point, I think one of the challenges within an infinite game is that it really brings everything into scope and it gets very complicated to reason about tradeoffs. It gets very messy.

The great thing about kind of narrowing a little bit is this is our goal that we're going to charge towards. We actually do much charging and writing software, but focus deliberately on implementing. It does help narrow things. It helps people make sure – It helps with alignment. So I do think in cases, really you want to have kind of a mix of both approaches, where for kind of tactically, once you do planning for each quarter or each year, you do want to have a clear goal that focuses and narrows and aligns. But when you're picking which clear goal that you want to pursue, you want to pick that goal in the broader context kind of bring in all these different constraints and kind of different horizons.

One of the metaphors I think about a lot, this iterative elimination tournament works critical to do well enough in a given round of the game to make it to the next round. But we also need to win the next round as well. Where if you do really poorly this round because you do like a massive investment in infrastructure, forget to build the product, you won't get to advance. You run out of money. You lose the customers or whatnot.

But, conversely, if you've never invested in infrastructure, at a future round, you will also just can't deliver and this like tension between how do we make sure that we do well enough this

© 2019 Software Engineering Daily

round to get to the next. How do we make sure that we go into the next round with the materials to support the people that we need to win that round as well has been one that's really powerful for planning in my experience.

[00:37:49] JM: You work on the foundation engineering team. That is Stripe's take on platform engineering. We've done some coverage of platform engineering at Uber, Netflix, Lyft, several other companies. What are some unique practices at Stripe of your platform engineering strategy?

[00:38:10] WL: I think there are a number of things that I think I'm quite proud of the team here at Stripe. I think one thing we do really well that I think it is rare is that we finish migrations. So, finishing a migration is not getting most people over and them like sending out like an email and like not talking about it. This is how some people try to pretend they finish the migration.

Finishing migration is driving up adaption completely and deleting or deprecating, decommissioning the old version entirely. So I think one of the things I'm most proud of that we've done here and keep doing is we actually finish migrations, and that I think to me just the full embodiment of like this consequent thinking, not like how do we do something really exciting, but how do we do it completely in a way where we get to deprecate what we have today. That's something I'm quite proud of.

Another thing that I think that we've done that is really important is that – And this is I think becoming more and more the norm in kind of Silicon Valley infrastructure organizations, but we really have been focused on the full kind of software model. We've not kind of gone down the classic kind of SRE model with a non-software skillset. We have people who are great SREs with a full software skillset at Stripe, but we haven't hired folks particularly without the software skillset.

This can be difficult and definitely can slow down hiring for a long time, because this mentality around reliability, efficiency, performance, that is kind of this SRE mindset is so critical to our success and to like any company's success at scale that it was hard for us to find people who had all of that and strong software as well.

But the benefit of doing that is it's really allowed us to approach each of these problems from a software perspective of how do we build the tooling, the automation, to automate this migration to prevent us from having any human involves. This has let us stay much leaner than kind of comparable organizations that I've seen in terms of kind of the people maintaining the systems on a day-to-day basis.

I think not naming names, but I once joined a company where when you were on-call for kind of the operations team, your phone would literally run out of power in your 12-hour shift. You'll get so many pages that your phone – You see the primary as like, "Please help. Can someone take over? My phone is out of charge," and that was just like a terrible experience.

Here, most of our people are running critical infrastructure, getting paged once a week or fewer. I think that's been something that requires as a constant thoughtful approach both from like a meticulous and kind of this reliability, sustainability, operability perspective. But also in terms of building like automation that does the work for us versus like triggering someone to come do the work on the systems.

[00:40:56] JM: It's telling about your career that I know many of the companies that you've worked at, and I can think of at least two where I can imagine you getting paged so many times that your phone runs out of battery. But these days, it's seems like many companies are able to get to a state where there is much less stress on the on-call rotations largely due to an improvement in technology, like cloud products have gotten more reliable. Kubernetes has been pretty helpful, continues delivery practices. A lot of cultural practices, like dev opsy kind of things. Do you still have outages at Stripe? Do people still get paged at night or has the infrastructure gotten hardened enough where people – It is a rare occurrence where you actually get woken up in the middle of the night?

[00:41:57] WL: It's just responding to this idea of like systems getting more reliable over time. I think what I've seen is actually a little bit like the NoSQL movement a decade ago when people are like, "Hey, SQL is too hard or doesn't work at scale. So we're going to build the systems that are so constrained. You can only do things that work at scale, kind of hard to work with." I think what's happened with a lot of these dynamic distributed systems is that not that the stuff's gotten more reliable, but they're actually designed acknowledging this stuff is not reliable at all

underneath the hood. Where you have like auto-scaling groups. You have kind of like load balancers that can route around like things that are not healthy. I think we've moved on to this model where instead of building on unreliable things as if they were reliable, we've just started building on unreliable things, acknowledging they're unreliable. That's quite a robust systems even though the components are not robust at all individually.

So I actually think – You mentioned cultural kind of practices, and I think that cultural practices I think best embodied by this kind of default engineering movement that's been developing or really what's made us more reliable, not actually anything working better. It's like just being honest about what does or doesn't work.

In terms of kind of outages at Stripe, we have something that I think is like one of our best kind of things that we've done internally in some time, which is we have like a really strong incident program. So, every time there's an incident, we do an after action review. We write it up. We have critical remediation that are like, "What would we need to do to make sure that if we had something similar happen again, it would not result in an incident? Then we use this to drive learning."

I think there's this idea that incidents are inherently bad, and making sure that incidents don't impact users, incidents that impact users are inherently bad and we need to make sure that doesn't happen. But incidents are actually one of the most powerful systems for learning. Really, I think of it as like an engine of learning that drives understanding of distributed systems is how they fail. If they're not failing, you don't have a distributed system, or you aren't changing it very much.

So, really, I actually think whenever we stop having enough incidents, we would change our classification to make sure we have more in terms of like how we think about this taxonomy of incidents, because we learn so much from them. We actually want to keep harvesting the learnings from these. But that said, there is a huge difference between an incident as something to learn from an incident that impacts our users and ensuring that that is not the case that we're having incidents impacting users is really like that the number one thing I think about. Number two – Security of our users is number one. Reliability, strong number two, kind of top of mind literally all the time.

[SPONSOR MESSAGE]

[00:44:40] JM: As a software engineer, chances are you've crossed paths with MongoDB at some point. Whether you're building an app for millions of users or just figuring out a side business. As the most popular non-relational database, MongoDB is intuitive and incredibly easy for development teams to use.

Now with MongoDB Atlas, you can take advantage of MongoDB's flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security, and high availability, and data recovery, and monitoring, and elastic scaling.

Try MongoDB Atlas today for free by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas as well as support Software Engineering Daily by checking out the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se.

Thank you to MongoDB for being a sponsor.

[INTERVIEW CONTINUED]

[00:46:03] JM: Why do engineering teams get less productive rather than more productive when they add engineers to the team?

[00:46:09] WL: There's kind of like two different reasons why teams get less productive. First, there's a classic, mythical man month. Fred Brooks, there's just communication overhead to having more people communicating on stuff.

The other bit is just training, like each person who joins take some period of time where you have to invest into them, coming up to speed on your technology, your approach, before they become like net positive. I think depending on how quickly you're growing, sometimes you're growing very quickly. This like hyper-growth scenarios and your training is not something you've

invested in. Actually, I think it's slower over time because you have more and more people who haven't finished ramping. It's so hard to ramp on your kind of complex to learn systems.

[00:46:52] JM: Does Stripe do anything unusual managing its code base? Do you have a mono repo or do you have lots of small repos?

[00:47:02] WL: Stripe is mostly a mono repo. There are a few things that kind of have spun out historically that we haven't fully brought back in. We find there's a lot of leverage for the mono repo, in and I think it's important to decouple this idea of like mono service to mono repo. Keeping the code together in one place allows you to do in one single commit kind of code modification across the entire corpus, which is a really powerful tool for automated re-factoring. Where if you have many –

[00:47:29] JM: Especially if you want to complete migrations.

[00:47:31] WL: Especially if you want to complete migrations. It's just so much simpler than having to coordinate many small pieces.

[00:47:37] JM: Dig into that in a little more detail. Explain why that's important.

[00:47:40] WL: So, if you think about a big change that we made, like two or three years ago, is we wanted to move from one kind of method, one client library for writing to our backend database to a new one, and the new one had this interesting property, which is like every query we're performing was statically types. So we could actually kind of make assertions about the types of data that would get written in. It could actually check that we're only writing in the types of data that we expected to through our –

[00:48:06] JM: And this is a library that was used by a bunch of different teams within the company.

[00:48:09] WL: And it was used by every single interaction with the database across many different, like hundreds or thousands of different kind of models that knew how to serialize themselves into the database. So being able to do that centrally from one team who wrote like a

– One of the coolest technologies out there and there's a great paper from Google about like [inaudible 00:48:29] at scale is just like rewriting the AST with software and you can actually do this very complicated migration in a relatively easy way with no errors if you kind of use these sort of tooling.

There's doing it, there's writing it, which is easy often. But then like rolling it out across many kind of different repositories can be very complicated, because you have to land review like hundreds of different patches. This is where – This is kind of a classic. Every company that's great at managing libraries or great at managing a mono repo, they're both like quite challenging in different ways. Stripe has really gone down the mono repo side of things versus the library kind of [inaudible 00:49:07].

[00:49:08] JM: Was that from the early days? It was at mono repo early days? So, this is one thing I found really fascinating, is like Google's mono repo. Facebook is mono repo. Stripe has gone for mono repo. But most people are not mono repo, and a lot of it seems to be – Well, I don't know why that is, but some of it certainly has to do with the fact that the tooling is really hard. I guess, the processes are kind of – Any reflections on why that is? Why is a mono repo not more popular throughout the software industry?

[00:49:38] WL: I think one of the advantages of the kind of the micro repo or whatever you want to call it model where you have many repositories is, one, you have very clear ownership where this is like my team service, my team repository. We manage it. Two, though, the tooling really does work better for smaller repositories. Until a company gets to be quite large, you usually can –

[00:50:00] JM: By tooling, you mean like the SaaS and PaaS services that are available in the public, not like your awesome, cool Google tool that they just wrote a paper about.

[00:50:09] WL: Yeah, that's exactly right. So, one of the challenges of dealing with a large mono repo is like git, HG. These tools actually don't work super well at a certain size where you end up having – Like Microsoft's done some really cool work around like how do you scale git. This is because –

Transcript

[00:50:25] JM: Right, because they migrated to git recently, like fairly recently.

[00:50:29] WL: But these tools that just aren't meant to be supporting like thousands of commits per day with like very large number of files. And your build infrastructure, your CICD, your containers. There are all these stuff that just doesn't work as well initially on repo that you have to invest in. Then once you've invested in it, you can make it work just as well and have all these advantages around kind of re-factoring very quickly. But the advantage of small repos is you just don't have to make that upfront investment and they will work very well for a long time to hit the scale points, which are slightly different. In that case, you're not going to have the scale point around like git or your build or Jenkins or something. Instead, you're going to have like how do I actually coordinate package upgrades and how do I deal with like upgrading hundreds or thousands of services. This is a really complicated problem to solve.

[00:51:17] JM: You get emails. You wake up to an email that's like, "This service has been updated. Please import the new package. Please run your unit tests to make sure it didn't break any of your dependencies," and you're like, "Oh God!"

[00:51:27] WL: I think a certain scale, what you have to do for either is not that different. It's just I think the smaller repository model lets you push off longer. If the vast majority of companies never get to scale, which I think is just kind of objectively true. Most companies are never going to have a thousand engineers. Pushing off this thing you might not ever need I think is like a pretty rational decision, but at a certain size, I think the tooling, they're able to build around mono repos is just so strong from a workflow perspective and it's hard to replicate.

I think a good example is like if you use Thrift GRPC, you these IDLs for each service. You update it, and a lot of companies have this workflow where you update it in this repo, you update the service, you update this client repo and then every client repo needs updated. Then like some of the client repos never update because the teams working on it are like doing something else. You just can never actually get anyone upgraded. If you have a mono repo, you can actually just do it all at once. So I think there's a huge power for larger organizations in a mono repo that is very hard to replicate in the larger proliferation.

[00:52:30] JM: Yeah, and this is the example you are giving before I interrupted you, is like you have this client library. When the client library got updated, because you have a mono repo, you can essentially do find and replace in the entire repo. Instead of me in the world with the small multi-repos waking up to an email that says, "Please import the new library and test it." I wake up to an email that says, "Your code was updated. It works just fine. Carry on with your day," and your faster service.

[00:53:03] WL: So, Netflix has actually, as I understand it, solved the latter problem you're describing. Although, they were a huge like innovative company that has the resources to solve it.

[00:53:13] JM: They're multi-repo, right?

[00:53:15] WL: They are, and they've it by automatically upgrading. Basically, automatically merging in library upgrades. But on the only way to do that is having immense confidence in your tests where they can actually merge in and automatically deploy this upgraded version for every service depending on it, because they have so much confidence in their tests. This is where going back to like chaos engineering, going back to like – If actually design stuff to fail frequently and tests at it, like properly fails. Then you can do things like this. But this is a very hard one property. Not many companies could say, "Hey, I'm going to upgrade hundred services and automatically deploy them, and I trust if they successfully deploy, it works." Very few companies can say that.

But in their case, they've done the work to preserve that property and their fleet of services, and that gives them the ability to build this automation or they can have kind of the advantages of the mono repo with like the kind the poly-repo strategy.

[00:54:12] JM: So, I think of Netflix as one of the most evangelical companies when it comes to platform engineering. They really talk a lot about their strategy to platform engineering. Just to go a little bit deeper on this because you do foundation engineering, which is Stripe's verson of platform engineering. Are there any inspirations you take from Netflix or do you know any notable ways in which your strategy around platform engineering differs from that of Netflix?

[00:54:40] WL: I think the thing that Netflix has been most prominently visible around is really their chaos engineering, and that something where we have adopted a number of their practices. I think if you start looking at the details, like there is this Spinnaker deployment tool that they built. It doesn't quite fit with our aesthetics around deployment. We're more kind of container oriented and that is more a little bit VM oriented.

But I think the design ideas behind much of their software is like very comparable to ours, and I think particularly when I think about what makes them really effective and I think what has made Stripe really effective is this willingness to go further into making the workflows great. So, I think something Stripe's been working on for the last like year and a half is this [inaudible 00:55:25] project growth actually added static typing to our Ruby codebase, which has allowed this huge amount of leverage.

I think having the ambition to actually be, "Hey, our software works this way, but can we actually make it meaningfully better?" in this is really difficult, but like highly impactful way, and then to actually execute on it. So I think having that ambition to not just accept the constraints that people are familiar with or to kind of – We started talking about this, like not just cargo cult. What good looks like? But actually having kind of like the courage to dream of like something being better than it is for the industry and then successfully execute it is what to me kind of represents like these best engineering organizations and what we're capable of when we are not kind of constrained by what we've seen before.

[00:56:14] JM: Is it a fascinating trend that we moved from popularization of Java type language to popularization of Ruby, Python, JavaScript, back to popularization of – Well, I guess now it's more like optional typing. It's like choose your own adventure typescript. Sorbet is the Ruby thing. I mean, why do you think that is? I guess optional typing is just the best of both worlds.

[00:56:43] WL: Well, it's kind of interesting, and just as a quick thought. The other interesting thing happening at like web assembly is like now we're moving into strongly type languages again, where like web assembly kind of requires –

[00:56:53] JM: And memory constrained. It's like C.

SED 851

[00:56:55] WL: It's like embedded programming or something all over again, right? So, there is like a cyclical major to this. I think the thing that companies like about optional typing is that it allows you to only pay the costs when you get the value. I think a lot of companies early on just start trying to run very quickly to build the their MVP to build to add features as quickly as possible. They don't have a lot of the problems that typing solves yet. As codebases get more complicated, not having typing means it's very hard to reason about changes. It's very hard to detect like certain categories of bugs. Strong typing illuminates entire categories of errors from your software entirely. You can still have other types of errors, but entire categories are just like eliminated can no longer try to call the string function on an end. This is just like really powerful in terms of reducing the ways that you fail.

But early on, your code is simple enough that you don't run into those often anyway. You have so few people working on it that you don't run into them often anyway, because the team has a high-shared understanding of how the software works. So I think why companies like optional typing get the full value of the dynamic typing early on, but not the costs of static typing. I think the costs of static typing are a little bit overplayed, but languages are often like a little bit less joyful to write in, right?

But then later, you can still get the value by adding them in. So, I think it's purely just about this migration path where your choice then is like, "Hey, you have this giant Python codebase. You move from Python to Go or Java or something or you move from Python to typed Python, and that's just like the much lower lift. The risk of moving from Python to type Python is like zero-ish risk. Worst case, it doesn't work, but you've only lost effort. It's not like you have partially implemented stack that doesn't work at all. You can do it in place.

So, I think making migration safe is so safe and cheap is huge part of what makes like a successful infrastructure, more engineering organization or an unsuccessful one. I think that has been the biggest allure of these kind of gradual typing strategies.

[00:58:53] JM: Slack has changed software engineering. Tell me an usual way that Stripe uses Slack.

[00:59:01] WL: I feel like Stripe's use of Slack is fairly typical. I can think of two examples. One, I think we have at least 50 different hot dog emojis, which is changing software engineering for the different. But I don't know if this is for the better or for the worse, but you can do a lot with hot dog emojis I've learned in the last six months.

Maybe more aspirationally, something – For our incident program, we have this great set of tooling about how do we actually manage incidents, and we integrate that like very tightly with Slack. You go to this website, it's like very short form you fill out, and then you start this incident. Automatically creates a new Slack room. Automatically invites the necessary people into that room. In that room, we have a bunch of Slack tools, like /IR- command where we can add postmortem items to review later.

[00:59:48] JM: Oh, that's great.

[00:59:49] WL: Can flag in new people. We can automatically send a notification internally. So, we just have all the tooling there, where it's not like you have to bounce out.

[00:59:57] JM: I want to buy that Slack pod now.

[00:59:59] WL: Yeah, we will consider selling it. I'm sure there's someone building this more widely, but this is like one of the cool things. All these companies have like all these amazing internal tools they build that they never generalize. Things like Sentry. Sentry, at one point, I believe was like an internal debug tool at Discuss. I think that's true, that then spins out.

[01:00:18] JM: Really?

[01:00:19] WL: Then you just have all these things -

[01:00:21] JM: Discuss, the WordPress common company?

[01:00:23] WL: Yeah, I'm pretty sure Sentry came out of Discuss.

[01:00:25] JM: Okay.

Transcript

[01:00:26] WL: But it's not just them. There's a call sorts of tools that spin up, but there's even more tools that haven't spun out that are equally cool that would be amazing if they did.

[01:00:35] JM: Talk about mono repos. I mean, Google has some great stuff for managing mono repos that I'd love to have access to.

[01:00:43] WL: Bazel is slowly trying to bring it to you, right? They're the diaspora of Google internal tools to open sources alive and well. I think Bazel is the open source version of Blaze. Kubernetes is kind of the open source version of Omega. There're all these things like slowly kind of creeping out. So if you just wait 5 to 15 years, I think your wish will come true. But it could come a little bit faster I suppose.

[01:01:08] JM: Speaking of Google, can you imagine a time when Stripe might go multi-cloud, or are you multi-cloud already? Do you go with BigQuery or manage TensorFlow or anything?

[01:01:18] WL: Long-term, I think I don't know if your reader of S1's, but I'm a skimmer of S1's.

[01:01:23] JM: I try to read them on my phone, and I can't find – Is there a good way to read them on your phone? I'm like save to pocket, and then it's like doesn't really work OpenRide. It doesn't save where [inaudible 01:01:34] in the S1. I need like a Kindle for S1's, basically.

[01:01:38] WL: Yeah, this is your new product idea. Small addressable market, but like very lucrative addressable market, I guess. As you read those, one of the interesting things that comes up is like people have to state all the risks. So like one of the risks that many companies talk about is they're like cloud contracts. So I think long-term, pretty much every company has to think about as they get more mature, like which clouds and how many. What are the kind of the core business continuity risks that you have to think about? Certainly, for Stipe, we also need to think about that.

[01:02:05] JM: You're talking for a cost perspective?

[01:02:08] WL: So if you if you look at some of the recent S1's, typically, there are cost that are contract size will be mentioned and which providers they're operating with. So the gold standard from an S1 perspective is to have like a very small commitment and to beyond like many, many clouds. If you're on enough different clouds, you might not even mention which clouds you're on, right? Where if you're only on one, you have to mention kind of the risk perspective which one.

Really, I think that is the gold standard for any company that is getting larger. The other thing that I think is really interesting to think about is that as data localization kind of becomes an increasingly important topic around the world, every cloud provider is not in every country you might want to operate in, and there's a lot of pressure I think to think about multi-cloud from that perspective as you look at entering more and more countries, which providers are actually there. So I think seeing the cloud photo providers expand to address the need of their users around data localization has been like to me one of the coolest things happening. In the cloud space in the last couple of years.

[01:03:07] JM: Right. Yeah. So you're going to have to like rewrites and Stripe services to write to Alibaba bucket storage instead of S3, for example.

[01:03:16] WL: There are a lot of cloud providers out there, Alibaba, Azure, TCP, Amazon. There just like more and more. As you start looking at some of these other markets, and China is definitely a great example of a market where – Actually, Amazon and Azure are both in China. I forget if Google is offhand.

[01:03:32] JM: Amazon is not anymore, I don't think. I think they had to pull out. Didn't they sell – Oh, maybe they sold old and somebody else is maintain it.

[01:03:39] WL: So, as of four years ago, which is the last time I deeply understood this AWS, but they operated in China, but it's a different company that is not technically AWS, but the code is shared, the interactions are shared, etc. But I think this is definitely an area where the lines are at times blurry and like a little bit confusing to understand.

[01:04:04] JM: Your book had a lot of material, and obviously we're not going to be able to get to even a fraction of it. So, certainly, for anybody who is in your book, I see a struggle to calcify how to think about engineering management. It's almost like your book is like the Gödel, Escher, Bach of engineering management. Is just like it seems like you cannot find the contours of this thing. Am I reading it correctly?

[01:04:38] WL: I think that's right. First, that's a comparison that I would like gratefully accept if anyone wants to make it. Yeah, I really do agree. I think engineering management is largely like making good decisions for this rapidly changing onset of folks that you're responsible for supporting within a larger set of folks who are responsible for partnering with, within the larger set of folks trying to do something with a business. So, I think just be the different shapes of the decisions, the different shapes of the people and their needs and like the company's needs and like all of it is so complicated that I think a lot of what I see is good management is that's like understand these different types of shapes. Let's figure out like a process that we found for finding good answers to them. There's not like one answer. There's not even like 50 answers, there're kind of infinite different approaches. It's by having great fundamentals and great kind of values of how we want to make decisions that we can find the appropriate solution to each one.

I'm a strong believer that if you understand the constraints, finding the solution is usually very simple, but it's often very hard to understand the constraints. So, to me, a lot of the book is about how do we think through what we really are trying to accomplish and the things that make that hard clearly, and then we can find an answer that is like fairly straightforward. But if we don't understand our own reality in an honest, authentic way, we can't actually make solutions that work.

[01:06:08] JM: Last question; if you had to leave Stripe today and start your own business, what would it be?

[01:06:15] WL: So, four years ago, my answer to this question was a kind of a geopolitically shorted database, where all this data localization, like regulation coming out, is meaning almost every company has to solve this data locality problem in their own way. Imagine what it would be like if you had a database you could just plug into that with shard and re-shard all of your data automatically according to the shifting geopolitical requirements. Every company, this is

like – So one of the things Stripe thinks about is how do we increase the GDP of internet, and part of that is like making it easier for small companies to compete effectively with larger companies. This is to me like this core thing that's going to make it very hard for small startups today to compete with like a Facebook or Google, a Microsoft.

[01:07:03] JM: Regulatory capture.

[01:07:04] WL: They just can't invest the legal and technical cost to actually solve for this. This is where GDPR seemingly actually not that onerous for the largest companies that was meant to regulate, but is very onerous for any of the smaller companies that are bearing a lot of the brunt with very few resources.

So, I really think that there's something very powerful here that could be quite important to make it easier for folks to get online, run their business and stay online in more countries overtime.

[01:07:36] JM: You said that was four years ago.

[01:07:37] WL: Yeah, the thing I think about more recently is very small, but systems dynamics is kind of this like study of kind of systems thinking what not and like how do you actually can model like real-world situations in a useful way. That the tooling for systems thinking is pretty bad, honestly. There's a couple of companies that are quite small that have been kind of updating their software for some time. Originally built 15, 20 years ago, and typically the per head costs of these licenses is \$3,000, \$4,000.

[01:08:08] JM: What does this software do?

[01:08:10] WL: It lets you design and run models. A good comparable I think could be like Omingraffle, which is a totally different software, but that's kind of the complexity of the software. It's like Omnigraffle though, most expensive software I've bought that's not an operating system. I think it's like \$200, \$300 to buy Omnigraffle now, which is a lot, but I think really well worth it.

But then these like modeling softwares is like an order of magnitude more, and like none of the software is like kind of native look and feel, like all kind of like custom, like GUIs components

built from like literally like two decades ago plus. I really you just love to build like a modern version of that. It's usable and priced like a delightful, like two digit price. Not like – I don't know, four digit price.

[01:08:55] JM: Cool. That is a software category I know nothing about, so I'll have to look into it.

[01:09:00] WL: This is why I'm not building it. It's a software category that doesn't exist, but I like it to exist.

[01:09:04] JM: Except in the world of - What, Omnigraffle?

[01:09:07] WL: No. So the software here would be like Stella. It is kind of a good example of it.

[01:09:12] JM: Stella. What is that? Stella.com?

[01:09:15] WL: I forgot the name of the company that does it.

[01:09:15] JM: Okay. All right, Stella, and the systems management, whatever.

[01:09:20] WL: Yeah. System thinking.

[01:09:21] JM: Systems thinking. Okay. All right.

[01:09:23] WL: It's really cool stuff, but mostly I think used by researchers and folks in academia at this point.

[01:09:29] JM: Okay. Well, Will Larson, thanks for writing a great book. Thanks for coming on the show.

[01:09:33] WL: Thank you so much for having me. This is great.

[END OF INTERVIEW]

[01:09:38] JM: GoCD is a continuous delivery tool created by Thoughtworks. It's open source, it's free to use and GoCD recently launched a test drive service that makes it easier than ever to try out GoCD. You can go to gocd.org/test-drive-gocd.

If you've been wondering about what continuous delivery tool you should use for your cloud native software, GoCD is worth checking out, and now it's easier than ever to just try it out and see if this looks like something that you would want.

Just go to gocd.org/test-drive-gocd and find out how GoCD fits your workflow.

GoCD has support for Kubernetes and it was built with the learnings of the Thoughtworks engineering team. If you want to try it out, go to gocd.org/test-drive-gocd.

[END]