EPISODE 919

[INTRODUCTION]

[00:00:00] JM: Software engineering is a new field. There are theories about how we should be building our systems, but these theories might change overtime. The same is true for engineering management. There are many successful examples of companies scaling with the same management hierarchies that have been pioneered by Microsoft and Google.

But since everyone knows that those same techniques work, they get continually copied. Of course, that makes some sense, because there's tremendous risk in pioneering a brand-new management structure. There have been examples of successful software companies that have suffered tremendously, because they decided to try an exciting new management structure.

New software is much easier and safer to try than revolutionary new management structures. New software is easier to try in a company with a strong engineering team also, because that engineering team is equipped to actually assess the new software and figure out if it's solving a problem rather than just introducing something new and flashy.

Uma Chingunde is an engineering manager at Stripe on the compute team. Uma has worked in management for a decade and has worked in virtualization on infrastructure for even longer than that. Uma joins the show to give her perspective on management of engineers, as well as management of compute infrastructure. We discussed some timeless principal of engineering management as well as contemporary ideas around virtualization and compute.

[SPONSOR MESSAGE]

[00:01:42] JM: When you listen to Spotify, or read the New York Times, or order lunch on Grubhub, you get a pretty fantastic online experience, but that's not an easy thing to pull off, because behind-the-scenes, these businesses have to handle millions of visitors. They have to update their inventory or the latest news in an instant and ward off the many scary security threats of the internet. So how do they do it? They use Fastly.

[INTERVIEW]

[00:02:56] JM: Uma, welcome to Software Engineering Daily.

[00:02:58] UC: Hi, thank you for having me.

[00:03:00] JM: It's great to have you. We had a nice conversation last time I met you at Stripe, and I'm looking forward to getting into more details around your beliefs around software engineering management, and infrastructure, and engineering at Stripe. So software engineering is a very new field. There is much unexplored territory in how software can be built.

But the way that companies get modeled is often by copying the management systems that have worked previously. In some sense, this is a great strategy, because if you've got a product that works, you could just copy the management style of Microsoft, or of Google. Is there anything wrong with that strategy? Is there anything wrong with just copying the management structures if you have a product that's already working?

[00:03:52] UC: This is a great question, by the way. It's actually one I think about a lot. I have a kind of nuanced answer. One is there isn't anything wrong. In fact, you do want to actually consciously imitate the better things. When I say there are some things that I have kind of observed that seemed to work are organizational hierarchies.

I think particularly more recently, a lot of companies have kind of played around with not having teams or having very fluid dynamic structures. In my experience, that tends to work with smaller

teams, but doesn't really scale well. So you see most companies eventually kind of converge into a more hierarchical organizational structure. That seems to work.

On the other hand, I think some things which don't work and I think companies have wisely probably abandoned are the way software itself is released, right? So when I started my career, I was working at VMware and they were in the business of releasing on-prem software. So it was very waterfall-style, like your multi-year released. That just doesn't work in a SaaS-based business.

I think you'll see very few companies that are in a SaaS model continue to use that release model, and a lot of things associated with that, for example, separate QA teams are less and less common now. Separate kind of release processes are less and less common now. Kind of the gating of releases is much less common and you're really in a world of continuous release. That's my view.

[00:05:26] JM: One area where it's hard to know to what extent we should be copying is in the hiring process. So you know that if you copy the Microsoft style of hiring or the Google style of hiring, it's going to work to some extent. You are going to hire engineers eventually. But there's not a really good way to test if that's an ideal strategy, unless you do A-B testing for your hiring process, but then that's really hard, because you have a very low sample size. What's the process for developing a productive engineering hiring process?

[00:06:12] UC: Okay, and a really good question. I like to think of it as while you can't really do A-B testing, what you can do is really gather data and maintain data at every step of the process. So where are you losing candidates, for example? What is your close rate of candidates once they get to offer?

Depending on like your process, you can actually ask candidates for their feedback, right? The best interview process in my opinion is one where even when you don't result in offer, the candidate likes the process. I think that is where you get through with constant iteration. A few things that I've seen that have evolved over the last decade are things like whiteboard coding, for instance, right?

I think that's just like a simple iteration of listening to candidates, listening to what folks like and turning it around and being like, "Okay, what are you actually trying to gain from a coding interview?" Then saying, "Okay, what we're trying to test is people's coding skills and do we need a whiteboard to test those skills, or can we actually let people bring in their own laptops and code out a problem?" So you've seen this shift happen. I think it's become more and more common across the industry.

Another example is kind of where things have shifted pretty dramatically are puzzle questions, right? There was this whole era where puzzle of various types, such as like manhole covers and why they're around, and things have really evolved there. I think just going back to what is it that you're actually trying to gauge? What is it that you actually need off the person that you're hiring? Then iterating all and consistently getting feedback and listening to your, essentially, users in this scenario and making a better experience for them I think is the way to go.

[00:08:00] JM: The manhole covers class of questions. I remember this period of time where people love to talk about this like it was a sign that these companies were doing something distinctly correct. Everybody was celebrating the manhole covers question, as if this was like a breakthrough in how to vet people for engineering roles. It's just funny. If something stands out as novel and a company is doing it, we can have this tendency to believe this company has had some sort of breakthrough. How else would they have come to such a crazy idea of asking people about the shape of manhole covers other than the fact that this is actually a magical panacea for solving your hiring problems?

[00:08:48] UC: Which is why I think it survived for so long, right? I think it was like the sheer novelty of it. I think it also appealed to people for good reasons, which was like, "Oh! We're actually trying to gain your ability to solve a puzzle in an intuitive way versus something else."

[00:09:06] JM: That's true. Yeah. Very good intent.

[00:09:08] UC: Yeah, it resonates with people, like, "Oh! You're actually looking for something core and fundamental versus some other label." So there were good reasons for it, and it survived for good reasons. But there are so many factors that influence hiring. For example, the market has shifted so dramatically in the last decade where your ratio of jobs to candidates has

really turned around. I think a lot of those things make a big difference in the way interview processes work.

[00:09:41] JM: The idea of copying what is working in the industry or what people are talking about in the industry also happens with technology decisions. So some engineers today feel like there is an inherent benefit in picking up Kubernetes, or going to microservices.

On the other hand, sometimes hype is beneficial, because it pushes you towards the new technologies. What's the right measure of a adjusting to the hype of trying out new technologies versus using the ones that you know do the job?

[00:10:24] UC: I think deeply understanding what your use case is. Really trying to understand what is it that you are building as a company, as a team. Making sure that I think a lot of people gets weighed in the hype when there's sometimes like a missing element of kind of grounding in what they are building and where they're trying to go.

If you don't know what your building or where you're trying to go or what kind of essentially your tech stack mission is and you don't have these grounding principles, then people tend to get distracted and kind of clutch at straws. Sometimes those become like the most hyped technology of the day.

I think one way that I like to think of it is what is it that is really core to your business? What is it that's really core to your tech stack, and improving that, and then using what's available to make that choice. There's a really good, I think, podcast or blog post by Intercom around build less software. I think that really resonates with me.

I think where you want to be careful is you want to make the right tradeoffs. You do want to optimize and develop for the new tools, because that opens up, that can potentially just like unlike huge gains for you. There is no benefit, for example, if you're doing exactly what Kubernetes provides trying to build your own internal distributed scheduler. There's literally no benefit at all.

So if what it provides is what you need, then yes, you should do that, because then it unlocks all these people that already have these expertise that can then come in and start working on you and can be productive from the beginning.

But the other example is when things don't work is, "Oh! You don't quite understand whether you need microservices or whether your monolith is still good enough for your server." But you decide that, "Oh! Microservices are the way to go," and you don't understand the tradeoffs and you go ahead and start breaking things up.

I think the easiest way to actually think about this is understand why you're doing something, and then de-risk to the extent possible. I'm a huge believer in prototyping early and often and really like testing things out rigorously in a smaller environment before adapting any large migrations or any wholesale migrations.

[00:12:51] JM: The run less software idea from Intercom, we had an interview with Rich about that idea. My recollection of that idea is it's in some ways the expansion of the idea of having blessed languages. So at Google, Google in the early days decide, "Okay, everybody was trying different languages." Some people were using Pearl. Some people were using Java.

Then eventually they just decide, "Okay, we're only going to use these languages. If you want to go outside of these languages, you need to have a really good reason, because we just need to have a little bit more control. We need to have a little bit more constraint around what languages we're running across our stack."

The way I saw the run less software idea is it's an expansion of that idea where you expand it to tools and cloud services where you say, "If we are going to run a relational database, we're using Amazon RDS and we're using PostgreS, and that's it." If you need to use MySQL, you better have a really good reason for it.

I can see that being really beneficial if you're talking about well-defined things like relational databases, non-relational databases. You need a compute unit. You're using a container. It's deploying to our internal PaaS. Take it or leave it. That's what you got to do if you're working here.

But it gets more complex if you start to talk about things like Redshift, or BigQuery. Do you start to mandate your data warehousing tools? To what extent should we have these kinds of run less software limitations and what are the good standards to impose in this world of so many different tools that are available?

[00:14:47] UC: I think it really depends on – Again, it kind of really goes back to your core stack and like what it is that you are trying to optimize for. So a few things to kind of think of there is what are you trying to build? One of the things I like in the Intercom post is what are your key differentiators and what is like – What you're trying to build more of is like your key differentiator. So you build that in-house, and what you try to buy off the shelf is something that is not your key differentiator. So I think like that's one good metric to use.

Another one that you kind of also want to balance is you don't want to make your rules so rigid that people cannot innovate. Because then you're constraining future products, or future development, because it's not what's already core, what's your core business today. But you might be stifling what is potentially the company's core business in the future.

So you kind of want to have a balance where the majority of your infrastructure team's users are directed towards a sort of golden pot and a sort of narrow kind of enclosure where everything is just easy for them. But then you want to have a flexibility where if someone does want to explore outside this, they're able to do this. You don't impose a high-cost on them experimenting. So you kind of want to have the balance. You don't want to have people introduce 10 new languages.

But if there is something that is really interesting and is useful for, say, a team within the company, you do want to allow them to experiment, validate it, prototype it and then probably say, "Okay, this is now proven its worth," and you bring it in. You want to reduce friction to trying out new things and then potentially adapting them or like letting them fail. The majority of users should have like a set of guidelines that should be easy to follow.

[00:16:43] JM: You're an engineering manager at Stripe. You are on the compute team. If I want to instantiate a unit of compute to run something at Stripe, what's my interface for doing that?

Am I deploying a VM? Is it a container? Is it deploying to an internal platform as a service? What is that deployment process look like and what am I deploying?

[00:17:08] UC: I think the answer to this question is actually more complicated than I would like to answer right now. So the short answer is we are aiming to provide an interface where it is as easy as possible. The longer answer is that we are not there yet. So, essentially, we have a combination of we're not at a point where we could actually provide a simple push button solution, but we have a combination of tools, a lot of which are homegrown, which are essentially common line tools that allow you to provision a new service.

Currently, it actually takes more steps that I would personally like, but that is something that I think is really valuable to try and improve. So this kind of goes back to the majority of the users should potentially just have like a couple of steps to deploy a new service. Then if there's like a power user that needs a very specific instance or a very specific EC2 instance, for example, and they actually care about the underlying hardware, we don't restrain them from using that and we actually allow them.

So where we are currently is that it's a bit of a mixed bag and what we're working on, it is actually a really good illustration of kind of separating your power users from the majority of your users, and we're trying to do that right now within our platform.

[SPONSOR MESSAGE]

[00:18:45] JM: This November, the O'Reilly Velocity and Software Architecture Conferences come together in Berlin to give you the latest insight on key trends for cloud native systems, dev ops, domain-driven design and microservices. Join 2,000 other engineers, developers and software architects for networking an essential training on how to design, build and manage resilient systems.

I've been going to O'Reilly Conferences for the last four years ever since I started Software Engineering Daily, and they're a great way to learn. They're a great way to have some food, network with people. On November 4th through 7th in Berlin, the Velocity and Software

Architecture Conferences from O'Reilly are coming together, and you can get 20% off your ticket by going to oreilly.com/sedaily and entering discount code SE20.

O'Reilly Conferences are a great place to learn about microservices, domain-driven design, management, software frameworks, cloud services. There are lots of opportunities to learn and get better and get yourself to a new job altogether if that's what you're looking for.

I've met some great people at O'Reilly conferences, and you can attend an O'Reilly Conference by going to oreilly.com/sedaily. Find out more about the Velocity and Software Architecture Conferences, and perhaps go to Berlin. These conferences are highly educational, and your company might pay for it if you ask your manager. But whether they do or not, you can get 20% off by going to oreilly.com/sedaily. Use promo code SE20 and get that ticket.

Thank you to O'Reilly for supporting us since the beginning with passes to your conferences, and thanks for producing so much great material about software engineering.

[INTERVIEW CONTINUED]

[00:20:47] JM: A company start to build this type of command lines, because I mean to me it makes complete sense for a company of Stripe's size to have done this. But like I think about Airbnb, for example. I think of Airbnb as kind of a contemporary of Stripe in terms of the company's maturity and how much traction the company's had, the quality of engineers they've been able to hire.

I have no idea if Airbnb has an internal PaaS or not. But I guess I'm wondering, for whom does it make sense to build this level of internal customization versus the kind of company that's just going to say, "Look, our interface is AWS. If you're an engineer here, you are working in the AWS console. That's what you do." Who should build their own internal deeply technical systems like these?

[00:21:41] UC: I think, again, there's no simple answer to this. For example, some of it actually depends on the timing of the company, right? When was your company started? There're things,

there are decisions, for example, that we have ourselves made that maybe we would not make if we were starting out today.

So, to me, the intersection is what is your company size? What are you building? But also like what's already available to you? For instance, if you were starting from scratch today, we would probably start using Kubernetes from the beginning and our architecture would look completely different. AWS has already provided far more sophisticated tools that may be some of our kind of like tools that we have built on AWS would not need to be built if we were building them today.

On the other hand, maybe when you get to a much larger size, you kind of get to this kind of the extreme end where you actually start moving off the cloud completely. Then you actually need to build an entire set of tools from scratch. That's kind of like the scale at which you're really large, like a Google or a Facebook.

So I think there is never an easy answer, and I think the same company will actually make different decisions based on its size, its time of creation and its tech stack. So it really has to be a decision made at that with those factors in mind, and within the lifetime of a company, you will actually revisit those decisions many, many times.

So I think an actual problem in my mind is if you make a decision and then you fail to realize when the decision is outdated. As an example, when I worked at a previous – A different company, much larger. They had a completely opposite view to open source software. They really did not believe in it at all, because it was shipping on-prem software.

But the downside of that was that we literally rebuild very, very basic tools internally over and over again. As an example, like the simplest example is that we had something like Splunk that we had rebuilt from scratch internally. When you think of things like that, just investing that much manpower into things over and over again, I think that is kind of like – That's where you want to reexamine your rules. Is it so terrible to actually like buy software externally or do we have to build everything ourselves? Really to me, it's more like you have guidelines and then you keep revisiting your guidelines over and over again versus never changing them.

[00:24:27] JM: When we last spoke, you mentioned that you are working on something-something service mesh. That was I think verbatim what you said.

[00:24:34] UC: Yes.

[00:24:35] JM: That was obviously funny to me because there's a lot of people in the world of software right now that are working on something-something service mesh, which is about how they would describe it.

[00:24:47] UC: Yup.

[00:24:48] JM: Describe the ideas around service mesh. First of all, why is this abstraction useful and why is it kind of a moving target? Why is the way that you think about service mesh today something-something service mesh?

[00:25:03] UC: Yeah. I want to first clarify. I did not actually something-something service mesh. I actually said something-something Kubernetes, because even in a joke, actually, everyone in our team has actually moved on. We don't even refer to it as a service mesh anymore, because, one, that term has gotten overloaded to the point of actually being meaningless. It doesn't really mean a lot anymore in my opinion to the point where you ask different people and you get a different answer, and it isn't really –

[00:25:32] JM: This is like the microservices version of big data. Nobody uses the term big data anymore.

[00:25:38] UC: Exactly, or cloud, or where anything, like literally anything can be overloaded. So we don't use that term internally anymore. I think what we're trying to really focus on is what we're using is we use Envoy and we use it as a proxy. We use Envoy proxy. We're using it for a very specific set of scenarios, and that's very specific to us.

That's how we like to think of our tools, where it's like less about the terminology and more about exactly a scenario that we are trying to solve for that's very specific to us. This is the tool. These are its benefits. This is our problem. This is what we're using the tool for. I think speaking

about like technologies in those terms is far more useful than the buzz word bingo kind of technology usage.

[00:26:31] JM: Yeah. I'm definitely hearing a theme in what you're saying, which is look for solutions to specific problems. I think this is definitely something that has come clearer to me overtime as I've kind of realized that the way that companies actually end up making money is that they solve problems for customers. That is who – People only pay money for software when it solves a problem for them. Then if you're solving problems for customers, that very easily boils down to you need to solve your own problems as well, because you're always going to have problems. Therefore you're not looking for fancy new things to do engineering diagrams around. You're just looking for solutions.

[00:27:19] UC: Yup. I think that's one of the kind of key things for infrastructure teams as well. So it's literally like what is it that your users need? In this case, your users are the other engineers that are using your services. What are their problems and what are you trying to solve for that are their problems?

If you use that user focus mindset, that's when you're likely to be successful, versus if you go down the kind of adapting new technology or adapting new things without a specific problem, then you're likely to get very deeply into trouble.

[00:27:58] JM: You've worked in virtualization infrastructure for more than a decade. What are your predictions for how virtualization software is going to be impacting organizations for the next five years?

[00:28:13] UC: That's a really good question. So I think I could be completely wrong in this. I think that the big changes have actually already happened in my view, which is the big change that I observed was where you moved from a world where you had people – So when you were like starting a startup in the 2000s, you would literally go out and buy a Sun server, something, some version of Solaris. You have to provision it, all of that.

Then VMware came along and made it all about VMs. So people moved. Sun died, and then you move to a world where you would like buy VMware software and provision it on Intel

hardware and then have your own data centers, and that's typically what most people's data centers look like.

But currently, the abstraction has happened to a point where very few people even have to think about the underlying physical hardware. To me, that is like the biggest shift that virtualization has made possible in the last two decades. I think once you've gotten to that shift, the abstraction is built so well that I think it's going to – We're already in a world where anyone starting a company today is unlikely to provision their own hardware, unless they have like very specific reasons. We're already in a world where the default is to pick from a handful of cloud providers.

I think that shift is where I see has already happened. To me, that has been like the biggest shift. I think what I expect to continue happening is you'll continue to see more and more of this where there are still a lot of companies that exist that are kind of in the service of those on-prem data centers that are selling software, primarily those.

Those will kind of consolidate overtime. You will have only a handful, maybe even just like three or four major cloud providers that will exist. The other end, you'll have the really large companies that, for their own reasons, like Google or Facebook, or similar, will have their own data centers, but will only do so because they're an extremely large scale.

Then everyone else will kind of exist in some of these world, and maybe there'll be small niche hybrid world in between. But in my view, the abstraction will just kind of continue rising. So we're still a point where people still actually think about EC2 instances and the underlying database and similar. But all the cloud providers are in their own ways building on top of that. So it's kind of you're getting closer more and more to workflows and what users need versus having to think of the underlying software that's like in terms of the hardware. So I think the abstraction and building the layers on top will just continue, but the biggest shift in my mind has been this, this kind of complete shift where you don't even think about your hardware.

[00:31:21] JM: As the requirements for compute infrastructure have become more demanding for data engineering teams, machine learning teams, how have you seen the requirements for

spinning up infrastructure diverge between the data teams and the service-based teams? Like just the microservices teams.

[00:31:47] UC: Can you say more about what you mean about the question?

[00:31:50] JM: If I'm doing data engineering workloads, at my understanding is I need like a GPU or I need like a bigger machine. I'm going to need more RAM to fit all the data on a single node, or maybe I'm just using Redshift, and that's abstracted away from me. So I don't know. It depends. I'd love to hear your perspective on that. But if I'm just spinning up a container that's going to be talking to a database and doing a little bit of processing and then returning that result to whoever requested something, the requirements are much more lightweight.

I'm just wondering, if you've seen in your time at Stripe a divergence between – A growing divergence between what the data teams need out of their compute infrastructure and what just random services need.

[00:32:38] UC: Yes. I think it's less because we're structured in a way that we're like further below all the other teams. We do see this, but I think we are still not at a point, or rather I have not been here long enough to see the sharp divergence. I think what is clear though is that, yes, we have – There are certain workloads that have very specific requirements, and that has been increasing. I think, for instance, are really large workload that need a very specific instances of our cloud provider, for instance, to the extend where we have to like request specific versions of things.

But I think another key thing is just the sheer cost of running some of these workloads and that adds up overtime. But I don't think at Stripe itself we are structured necessarily in a way where we've not gotten to a point where it's completely like black and white. I think our workloads are like still very much all mixed up together, if that makes sense.

[00:33:44] JM: Yeah, definitely. Tell me more about how cost management affects engineering decisions at Stripe.

[00:33:52] UC: We are still early on in our journey, where I think we still don't try to make decisions based on pure cost, because I think for any company that's in the bay area, the highest cost tends to be engineering cost, like the actual individuals versus the cost of your cloud provider or similar.

The way at Stripe we try to think of this, and there's actually a different team which I really like, because you have an entirely separate team that's thinking about this problem, is you try to actually provide the data to everyone about how much expense they are generating, or like how expensive certain things are.

So we're just trying to gatekeep or police different teams. You actually try to provide them the data with which then they in turn can make their own decisions, and that's the kind of general guidelines. So the first step in all of these is actually trying to figure out where you're spending the most. Then based on that, you can then make decisions versus trying to make a decision about we're going to spend less and like go find the savings, right? Kind of goes back to knowing.

I think my overall team and all of these is really understanding what your problem is, first, and then trying to solve it. Because if you don't know where you're spending the money or how you're spending it or what workloads are the most expensive, you might not be solving the right problem.

[00:35:22] JM: As an engineering manager on the compute team, what does your day-to-day work look like? Tell me about a day in the life of working on the compute team.

[00:35:37] UC: So I think my work looks a lot like most managers at Stripe, especially on the infra side. I manage a team that's actually for sub-teams, and currently I'm trying to hire for one of those teams. So hire a manager. So a lot of my time typically goes in interviews and outreach and I actually spend a fair amount of time trying to actually reach out to candidates directly myself. So there's a kind of good amount of time in spend hiring.

Depending on like where we are in our planning cycle, there's some amount of time that goes in planning, and planning off means having a lot of conversations with different people across the

company and trying to figure out our priorities and what different people need us to do and what we need other teams to do and kind of trying to mesh those things together.

So there's like the process and the planning and just kind of like the actual running of the teams. Then I think I also spend a fair amount of time on the people side and trying to actually help my team grow and develop and actually like hopefully get better at their careers. So I think that's roughly where I would say is my time split through the week.

[00:36:59] JM: In what ways does engineering management at Stripe differ from other companies that you've encountered?

[00:37:06] UC: So this is only my third job as an engineering manager. I think the biggest difference to me is the kind of team that I manage compared to my previous two roles. In both my previous roles, I was managing product teams, while they were like products for infrastructure, versus here, I manage an infrastructure team within a product organization. So that's like a key difference, because our users are fundamentally different. So at Stripe, my team's users are internal to stripe, versus in all my pervious roles, we were building products for external users, and that has been quite a big shift.

In terms of also where Stripe is, technically, it's pretty different. It's a SaaS-based company versus the pervious two companies were not. What I really like is the pace and the energy that working at Stripe brings that I haven't encountered previously. We're at this really exciting time where everything is just – We're like really shipping things incredibly fast and there's just this sheer air of just like optimism that I encounter every day, which is not as familiar to me.

I think it's really – I don't know if it's the same in other growing companies, but it's something that I really like about working at Stripe, which is you really feel like you're building something meaningful. You're building things that people are using. You get a lot of positive and quick feedback on the things that you're building. You see people actually using your products out in the wild. That to me is just like very, very gratifying.

[SPONSOR MESSAGE]

[00:38:50] JM: I remember the days when I went to an office. Every day, so much of my time was spent in commute. Once I was at the office, I had to spend time going to meeting rooms and walking to lunch, and there were so many ways in which office work takes away your ability to be productive. That's why remote work is awesome.

Remote work is more productive. It allows you to work anywhere. It allows you to be with your cats. I'm looking at my cats right now, but there's a reason why people still work fulltime in offices. Remote work can be isolating. That's why remote workers join an organization like X-Team.

X-Team is a community for developers. When you join X-Team, you join a community that will support you while allowing you to remain independent. X-Team will help you find work that you love for some of the top companies in the world. X-Team is trusted by companies like Twitter, Coinbase and Riot Games.

Go to x-team.com/sedaily to find out about X-Team and apply to join the company. If you use that link, X-Team will know that you came from listening to Software Engineering Daily, and that would mean that you listen to a podcast about software engineering in your spare time, which is a great sign, or maybe you're in an office listening to Software Engineering Daily. If that's the case, maybe you should check out x-team.com/sedaily and apply to work remotely for X-Team.

At X-Team, you can work from anywhere and experience a futuristic culture. Actually, I don't even know if I should be saying you work for X-Team. It might be more like you work with X-Team, because you become part of the community rather than working for X-Team, and you work for different companies. You work for Twitter, or Coinbase, or some other top company that has an interesting engineering stack. Except that you work remotely.

X-Team is a great option for someone who wants to work anywhere with top companies. Maintaining your independence, not tying yourself to an extremely long work engagement, which is the norm with these in-person companies, and you can check it out by going to x-team.com/sedaily.

Thanks to X-Team for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:41:25] JM: I saw a talk that you gave about career strategy. I looked through the slides and watched a little bit of the talk. There was one slide you showed that was about calendaring, like personal calendaring. I have a few friends who are engineering managers who have shared their calendars with me. On a rough week, it's just back-to-back everything. It looks tough. It looks like a lot of conversations with a lot of different people. It looks like there is no time between getting from conference room A to conference room B. Is that an anti-pattern? Back-to-back everything in your calendar?

[00:42:07] UC: I think the short answer is yes, it's an anti-pattern. But if you looked at my calendar, you would very much see the same today. So I completely agree that that's not how things should be. But for myself, I would be the first to confess that it's still very much a work in progress.

I think that many years ago I actually took this course where there's kind of like – I don't remember a lot about the course, which is a management course, but they kind of talked about being in two different modes. One is like red, where you're kind of like reacting. Essentially, like – That is the one I think of when my calendar is full back-to-back, where you're kind of essentially driven by external things and you're just like reacting to things and you're just continuing to do things based on essentially reaction. Whereas a green mode is when you're like pausing and when you actually stop and you're like planning and strategizing and optimizing and building forward versus fixing fires.

I think being in either mode for too long is unhealthy. So you always want to strive for a balance. I think this is where you kind of, what I've seen a lot of my colleagues do really effectively is actually use the calendaring intentionally. So you actually put up blocks of time where you actually schedule times where no one else can overbook, can book over it. You schedule particularly like downtime. You schedule things where you're like – You schedule time to think. You schedule time to plan. So you use the whole idea of like having a back-to-back calendar, but you turn it around and actually schedule time for those things.

I talked about this in my talk a bit as well, which is schedule time for learning. So I'm not there yet. I haven't actually been that great about doing these things, but I think those are absolutely the right way to actually like get on your calendar. You have to actually like block things out intentionally.

[00:44:09] JM: Yeah, I put these events on my calendar, these reoccurring events like drink three glasses of water, which it's like a 30-minute event that I have every day. Originally, I did it literally as a reminder to actually drink more water. But one thing I realized is by doing that, I actually was kind of blocking off time in my calendar, which gave me these nice breathers between when I would schedule things.

So if you're going to go back-to-back, maybe schedule almost none events or schedule like two-minute events that you can allocate 30 minutes to on your calendar. Anyway, terrible productivity tip from me.

[00:44:45] UC: No. That's actually great. Another very simple thing I do is all my meetings and either like the five-minute or like the 10-minute before. I think really just like questioning everything on your calendar. Does everything really need to exist? For instance, if you have a weekly meeting and you continue to have it, maybe not actually scheduling it to like never end. Having like set expiries so that you actually have to question whether you still need that weekly meeting with someone. I do an occasional thing where I'll just like the week before revisit literally everything on my calendar and just like start saying no to pretty much everything that is not absolutely necessary.

[00:45:27] JM: Right. Yeah, it's not like the Kubernetes scheduler where it's like you want it to be extremely reliable. You want to adhere to it as much as possible. This is like a sometimes effective scheduler.

[00:45:40] UC: Exactly. I think just recognizing that balance and making sure that you're actually like building buffer time is actually really important.

[00:45:48] JM: I'd agree with that. This was a talk that you gave on career strategy. Of all the things that you could have discussed; infrastructure, virtualization, payments, you chose career strategy. Why that topic?

[00:46:05] UC: That was just something that was top of mind for me. When the talk – When I was submitting the talk, which is something that was very much top of mind for me. One is I was – I talk about it in the introduction as well, which is it's something that was top of mind for me, which is I'm at a point in my career which is kind of very much like kind of inflection point, where there are multiple options open and I can choose there.

I was then, and still I'm at a point where I'm actually now supporting far more senior engineers than I have previously. So you hit this point in your career where you're kind of like looking at all the options around you and trying to decide what you want to do.

So to me that was actually like the most active problem on my mind at that point, and I thought it was one of the things where I had spent a lot of time thinking. So things I kind of like to talk about are things where I have actually encountered a problem and where I've spent a fair amount of time thinking. That's how I ended up proposing that particular talk.

[00:47:11] JM: One of the areas that you explore is the career ladder, and career ladders are useful in the sense that they can give an engineer a map for how to move up as an engineer. Here is how you go from entry level engineer, to SDE2, to SDE3 and so on. Here's how you go into management. Here's how to become a senior manager, principal engineer, etc.

The thing about career ladders is often times I see that the people who make it really far in an organization or the people who make really fast progress in an organization, at least in modern organizations, they're not the ones who follow the career ladder. They're the ones who invent something new within the company and then they get to start an entire business unit. They're leading that business unit. Their salary kind of takes on a different momentum, because it's no longer like really capped, because they're sort of leading an internal entrepreneurial venture.

So the idea of inventing something entirely, do you think we should have a separate track in the career ladder that encourages people to do that?

[00:48:21] UC: I think of it as – Actually, I don't know if all companies do this, but I've actually seen this pattern in multiple career ladders, and Stripe has it as well, where your growth is actually determined by the impact. Different stages in the career ladder basically demonstrates your impact.

So you start with like impact at the team, impact at the company – Sorry. Impact at the team, organization, Stripe-wide, or like industry-wide, and that kind of like defines your progression in the career ladder. That was actually true at VMware as well, where you're kind of like progression in the career ladder was kind of determined by your impact. Often the folks at the highest levels in the career ladder were people who had done what you described.

As an example, VMware had this title of principal engineer that was pretty much reserved for people that are kind of essentially built out the entire like new area of virtualization. The person who kind of developed VMotion, which was like a key technology was one of those principal engineers.

So I think that already exists. I'm not sure if it's really needs a parallel track. But I do think that there are often times when there are people who come along who don't necessarily fit in with that career ladder. I think that's where you kind of go back to the idea of like the majority of folks will fit into your existing career ladder. But what do you when someone comes along who is doing really great things but doesn't exactly fit into that? I think that's where your career ladder needs to maybe be a bit more flexible. So I wouldn't call it so much as a parallel track as maybe flexibility in the career ladder.

[00:50:05] JM: There's this book called The Alliance, by Reed Hoffman. I don't know if you've seen this book.

[00:50:11] UC: I have not read it. I have heard about it, but I haven't actually read it.

[00:50:15] JM: It's an interesting idea, which is that as a manager or a superior to somebody within a company, your relationship with your lower employees, the people who report to you,

should be that your primary relationship that you're developing with them is actually – Well, I could be misquoting the book.

But I believe that the primary relationship you're developing with this person is a potentially life-long relationship where you're going to be an ally to this person in their full scope of their career, and your default assumption should basically be that this person is going to be at the company for 18 to 24 months, and then they're going to go on to a different job.

Do you have any reflections on that idea?

[00:51:03] UC: I think that idea resonates me a lot. It's something that I personally believe in from both sides, which is where I think one is our careers are so fueled now, right? For instance, it's like no one stays at the same company for longer than like a few years. So careers are fueled. People change careers. Things change.

So in the end, what really happens is that to me it is a lot about the relationships you build along the way, and this goes for pretty much all of them, including the one of a manager and the person reporting to them. I think it is – To me, I encountered a success whenever I have a relationship with someone who's reporting to me, but that relationship is not tied to their particular job, where we have built that relationship where we can actually continue to work in multiple places together, for example, or we can continue to rely on each other outside of that one particular project or one particular role that we worked on together. I think that to me is like where you actually build a truly productive kind of, I guess, like the book says, alliance.

I would actually like to expand it more than even just the manager report relationship, right? This actually probably goes across the board. It's actually like this is true for peers. This is true for like multiple people in any organization. We actually see this a lot in Silicon Valley where groups of people kind of go on from company to company together and really like create a lot of really great things together.

[00:52:46] JM: Yeah, I'd agree with that. The intersection between programming and money, how will it change in the next 5 to 10 years?

[00:52:58] UC: This is a really interesting one and really open-ended. I think there are so many variables that's really going to be interesting. I want to assume that when you say the intersection between programming and money you mean kind of like what people get paid to program. Would that be correct?

[00:53:15] JM: I actually meant on the side of Stripe's business, like the API for payments. But the first one is interesting as well, like you're talking about salary compensation. Let's do salary compensation first.

[00:53:29] UC: Okay. I honestly have no idea about the salary compensation, because my answer there was going to be that there're just like way too many variables to actually like be able to make an accurate prediction. On this, for instance, there're so many things that are like policy or government of just like market forces that play into that that it's really hard to actually make a prediction. That was kind of going to be my non-answer to that question.

On the other side, I think that's actually even more interesting question around like programming and money. I think what I find fascinating is how much things have changed there. For instance, I started – When I first started traveling. This was like back when I was very young, in my early 20s. People still – Like you actually still went out and bought traveler's checks before, like international travel. This was only the tail end of credit cards were already very common. But credit cards often didn't work like outside of the country that you kind of were in and like your credit card would often get declined. So you'd either like actually exchange cash or do something like traveler's checks and then travel with it.

But with the extent to which this has kind of been abstracted like through programming is just to me really fascinating. Where kind of similar to virtualization where we've essentially added like these layers of essentially software on top of like currency that very few people actually have to handle hard cash these days.

As an example, I was traveling in Sweden over the summer, and Sweden is almost like completely a cashless society. It's actually really hard. I think I went my entire trip without encountering any currency, like literally. I was there for two weeks and I did not actually see Swedish currency. I believe they use Euros. But if you actually ask me, I would not be able to

answer, because I literally only used my standard American Visa credit card for the entire trip for like literally everything. There was one place where a train station had pay use restrooms and you actually had to use your credit card. It did not have a coin or any other like way of accepting payment.

So to me it's kind of similar to the way virtualization has worked, where you'll build these layers and that have just like made it smoother and smoother, and in very similar ways, right? When you're like dealing with hardware, you actually have to worry about the machines and like moving your software between machines. So here you think about countries as being those boundaries. Now with the way payments has advanced, like everything is like so transparent.

You actually encounter the underlying currency, and I think that's going to become far more widespread. To me, it's kind of like – Sweden to me was like a little bit of a glimpse of maybe what things will look like more commonly across the world.

[00:56:26] JM: One thing that I find kind of cool about – At a certain point in doing shows on Software Engineering Daily, the cryptocurrency boom happened. When that boom happened, I started reading a lot about it, because a bunch of listeners wanted to know more about cryptocurrency. So I did a bunch of shows and I read a bunch about it. Money is interesting, because when you go down the rabbit hole of what money actually is or what money represents to people, it can really turn a lot of beliefs that you have on their head, at least in my case. It was almost like – I don't want to call it like a religious experience, but it was like a very jarring experience, because for a long time I was like, "Yeah, dollars and these green things. Yeah, it's money. It's got value."

Are there any ways in which working – So this would be the last question. Are there any ways in which working with money at such a granular level have kind of restructured your beliefs about value or finance or anything that you held dear for a very long time?

[00:57:35] UC: I don't think it has really restructured my beliefs in a way, but I think the idea of Stripe's mission, and I think just the idea that access to opportunity can really change things for the better. I think that to me has really been a very kind of positive learning over the last couple of years, which is we're allowing people access to things which was like not possible before.

That to me was a pretty big shift in learning and kind of just like thinking about the way things work in the world.

[00:58:13] JM: Yeah. Even just the API surface. I mean, I found it transformative in how I think about software. So, yeah, I've been a big fan of Stripe.

Uma, thank you for coming on the show. It's been really fun talking to you.

[00:58:26] UC: Great! Thank you so much.

[END OF INTERVIEW]

[00:58:37] JM: Software Engineering Daily reaches 30,000 engineers every week day, and 250,000 engineers every month. If you'd like to sponsor Software Engineering Daily, send us an email, sponsor@softwareengineeringdaily.com. Reaching developers and technical audiences is not easy, and we've spent the last four years developing a trusted relationship with our audience. We don't accept every advertiser, because we work closely with our advertisers and we make sure that the product is something that can be useful to our listeners. Developers are always looking to save time and money, and developers are happy to purchase products that fulfill this goal.

You can send us an email at sponsor@softwareengineering.com even if you're just curious about sponsorships. You can feel free to send us an email with a variety of sponsorship packages and options.

Thanks for listening.

[END]