Life of An SRE: Life Beyond Google

ROBOT: You missed a page from the robot.

MP ENGLISH: Hello, and welcome to the final episode of season 2 of the Google SRE podcast. Or as we affectionately refer to it, the Prodcast. I'm your host, MP. And we have a very special episode in store today.

So through all of our previous episodes, we have had current Googlers join us to tell us a little bit about their life and their role at Google and their experiences as an SRE. And today, we have three people joining us who are former SREs at Google who have now gone off into different ventures. And we are going to hear about their experiences.

And joining me for that conversation, I'm very grateful for all of his work in putting this panel together, is my co-host today, Steve. Can you introduce yourself, Steve?

STEVE MCGHEE: Hi, MP. Happy to be here. I'm Steve McGhee. I kind of fit into the previous category as well, because I was an SRE for about 10 years. And then I left. I became an ex-Googler, a Xoogler. I joined a different company for a while, and I learned how to cloud. And then I discovered it was very difficult, so I came running back to Google.

No, I actually came back to Google to help customers learn how to build resilient and reliable systems on the cloud. That's my job now. So I'm a reliability advocate, which is a title that I just invented. It turns out you can just do that. And now I do stuff like this. It's pretty great.

MP ENGLISH: Thanks so much for joining us today, Steve, and for putting this all together. It was so helpful to have you get all of this organized for us.

STEVE MCGHEE: You got it.

MP ENGLISH: So let's go ahead and have our panelists introduce themselves.

CODY SMITH: Yeah, hi. My name is Cody Smith. I was at Google from 2004 to 2018. I started in a group called Cluster Ops and spent most of my time at Google working in Search.

And did some sort of org-level leadership as a production tech lead in SRE for a while, and then briefly was in cloud before leaving to co-found a startup working in energy, actually, called Camus Energy. I'm the CTO there now. We help electrical utilities decarbonize.

CARLA GEISSER: Hi. My name is Carla Geisser. I was a site reliability engineer at Google from 2004 until 2015. I worked primarily on large-scale storage systems that underlie lots and lots of Google's infrastructure. And after that, I spent a little while doing consulting in the federal government. And then I went to form a small incident response consulting company with a few other former Google SREs.

LAURA NOLAN: I'm Laura Nolan. I was at Google from 2013 to 2018, so I was a little bit of a later vintage than Carla and Cody. While I was there, I was an SRE mainly on ads, pipelines, and databases. We had this very big data warehouse thing called Mesa and a multi-homed pipeline called Photon. You can read papers about those.

Later on, I made a completely different move and went and worked on the network edge, which was sort of being-- the networks teams were being folded into SRE at the time. When I left Google, I worked for three years at Slack. You may have seen some of my incident blog posts that I wrote there during that time.

There, I worked mostly on service discovery, service mesh, and ingress load balancing. And I'm currently at a very small startup, very early stage, called Stanza. I'm an engineer there. We are making production predictable. And currently, we're working mainly on flow control and automation. Yeah. And I'm in Ireland.

MP ENGLISH: Thank you all for joining us today. Steve, why don't you give us the first question for the day?

STEVE MCGHEE: Sure. The internal joke about SRE is when you start realizing just how much stuff you learn inside of Google is like, is any of this going to be applicable if I ever choose to leave Google? And so I guess my first question is, was it?

I kind of know my own answer. And I think I know that, of course, things are transferable. But how different was it when you left Google? Did you find that your deep knowledge of Stubby was helpful?

LAURA NOLAN: Yeah. Leaving Google was a bit of a shock to the system. And I was only there for what, five or six years. So it wasn't like I didn't remember the outside world. But when I went in, things like Terraform were just getting started. And you come out and the world has changed.

I think a lot of the deeper, more theoretical stuff is perfectly applicable. So thinking about things like, how do I design my system to scale? How do I go about doing things like troubleshooting and analysis of things? All perfectly transferable.

But then obviously, it's the tooling that's the part that you have to come up to speed with. I think anyone who spends several years in a large company that does things its own way is going to have a bit of a learning curve on the outside.

CARLA GEISSER: The way I saw this framed by a friend recently was that the proper nouns are different when you leave Google, but the concepts are all the same. So there will be a thing that looks like a load balancer. There will be a thing that does rate limiting. There will be storage systems.

They'll all have different names, depending on what cloud provider you're using or what internal systems have been built bespoke inside of that organization. But the reasons those things exist and the reasons they fail will continue to be basically the same. And so that mapping was very helpful in every environment I've worked in.

CODY SMITH: Yeah, I would agree with that. I'm certainly not going to miss writing borgmon rule language. But I could write Prometheus configs instead and it would be very similar. The technology choice, I would say, is one of the bigger challenges, adjustments to make. You have to be able to look at some open source piece of software and figure out if it's a good idea to use it or not.

Whereas at Google, you're surrounded by people that have already looked at the options and made decisions, and there's sort of an emerging consensus from the community and within the company. And so you don't necessarily have that when you're out on your own.

CARLA GEISSER: Another thing I noticed after leaving Google is just the wild difference in scale. There's that vintage, at this point, I just want to serve five terabytes of data, video floating

around. And there are very few organizations where five terabytes of data is a small amount.

That was a joke at Google, but most companies just don't have that many computers. They don't have that much data. And so that difference changes a bunch of the stuff that you will have to deal with in the outside world.

LAURA NOLAN: It absolutely does. I think another thing that's really interesting about the outside world is you come out and you don't have that same uniformity that you get lower down the tech stack at Google. So at Google, you could rely on everything being [INAUDIBLE], right?

Whereas you come out and maybe you're dealing with Thrift and MessagePack and God knows what. And you don't have the Google three affordances of them having your [INAUDIBLE] and your [INAUDIBLE] and all of this kind of stuff. And that makes it harder, in some ways, to write tooling, because you don't have that sort of guaranteed control surface to latch onto.

I think that's why we're seeing such a big explosion of tooling around Kubernetes, because Kubernetes does give you a relatively uniform kind of control surface that you can build software according to and have it be relatively transferable organization to organization.

CODY SMITH: I will say, the general knowledge and experience working inside of Google is pretty valuable. The ability to work with data at scale makes you just kind of a magical wizard. Camus works with a lot of electrical utilities that have not Google scale data, but data that's big enough that they grapple with it.

So we show up and we say, oh, yeah. We'll just check that in BigQuery. And then you can do a query over all of your smart meter data for the last five years in 10 seconds, and it's eye-popping. That familiarity with the cloud tools goes a long way.

MP ENGLISH: I did actually want to dig in a little bit more about-- I guess to coin a term, this scale shock of moving from Google outside of Google.

CARLA GEISSER: I would say there was an experience once I had even before I left Google, when I was helping in the healthcare.gov war room, where someone came up to me and said oh, this must be the biggest computer system you've ever worked on. And I think there were maybe 500 computers, at most, involved in that system.

And it just was, compared to the systems that Googlers deal with on a day-to-day basis, it was nothing. But it was huge for the federal government. Even for a medium-sized company, that was a big software system. But for me, it was not very much. It was still complicated and broken in all sorts of amazing ways, but it was not big. There's a kind of a scale in complexity as well. That was something that struck me at Slack.

If you're changing something at the infrastructure level, or you're bringing in a new thing and you

want to onboard most of the existing services onto it, you can kind of actually just go and white glove them all, as opposed to having to go and run around and try and get teams to do this for you or build tooling. It actually becomes more economic to just go and make the changes yourself, because you're dealing with dozens or low hundreds rather than thousands of different services. So that's a kind of a difference in scale as well

CODY SMITH: Yeah, certainly during my time at Google, you never would have even fathomed having a service that had no replication, even within a cluster. Whereas now, we have deployments of our product for customers with, every single deployment has the same number of replicas of our front end, which is one. We've never needed more than one. It's never come up.

STEVE MCGHEE: So this brings us to our next question, which is, what do you love about your new job or your new role? Or when you left Google and you joined the next place, what were you excited about? And were you super happy to see some either piece of tech or culture, or maybe the snack tray? Like anything. What was it that made you smile in the new place?

CARLA GEISSER: After Google and after a few intermediary jobs, I formed a consulting company called Layer Aleph with a few other former Google SREs. And I love it, because it's the best parts of the early phase of an SRE project. We typically show up and work with teams that are trying to improve their production operations or trying to recover from a crisis. And we spend a very short amount of time with those people, in, the order of a handful of weeks, and try and help them work through their problems and figure out how they can improve their systems.

And then we write up our report and go on our merry way and hope that we were helpful. And so for me, that's the best part of my experience of being an SRE, is the learning, the quickly making connections with other engineers in an organization I've never seen before, learning how their systems work, and trying to figure out what's going on as quickly as we can. So it's been a lot of fun.

LAURA NOLAN: So I think the thing that I value most about where I currently am is, we're very small, which means that we can move fast. We all have high level of autonomy. We're working on a product. It's a reliability-focused product, but it's still a product, so there's a sort of a different element of creativity there as well.

Well, having said that, [LAUGHS] sometimes I miss the old days of debugging weird, gnarly production things. What Carla said there about some of the best parts of SRE being figuring out the connections between systems is definitely a thing that resonates. So instead, we're trying to build tools that will help people to do that.

CODY SMITH: Yeah, I guess my experience was kind of weird. I moved over to a completely different sector, working in energy with electrical utilities. I joke with people that working on Search, my product had a billion customers and I was surrounded by them all the time.

And now at Camus, we have six customers. I have a human relationship with all six of them. And working on decarbonization, especially right now, is just super rewarding. They have all these projects that are hung up on fairly simple things, like we need this battery to essentially talk to these transformers to figure out when to charge and discharge.

And that's it. It's a pretty simple technical problem, and solving it will allow them to deploy a 20-megawatt or a 50-megawatt battery and offset a lot of carbon emissions that would otherwise be drawing power from nearby power plants.

STEVE MCGHEE: When you got to your new job, I guess the next question in this line would be, so what did you do or add first? And whether it's like a piece of tech or a way of doing things or whatever. And specifically, with regards to reliability.

Like sure, you had to be onboarded and stuff like that. But what did you find that you either found as low-hanging fruit that you went after or something substantial that you brought to the table that wasn't in your new environment that you wanted to achieve first?

LAURA NOLAN: The first thing that I did, my first job after Google, my team didn't have a weekly production meeting. And this is a thing that I think is a really valuable piece of culture and process. So just the idea of getting together once a week and having a meeting that is solely focused on what has happened in production, what are on-calls struggling with, what are we planning on doing in the next few days, what's risky, what's going well?

That was something new, and that was something that I think was very helpful through that tenure. And then the technically, the first thing was putting in circuit-breaking and throttling in a few strategic places that really needed it.

STEVE MCGHEE: Can you define those last two terms for anyone who might be unfamiliar? Or even call out a tool or a method or anything?

LAURA NOLAN: Sure. Circuit breaking is basically when service A is talking to service B. And service A notices that service B is-- or perhaps a particular instance of service B is implementation dependent, is consistently returning bad results or is consistently slow. What you can do there is you can basically back off for a little while.

The idea here is that if you have a time-limited overload, you can stop making it worse by sending more results. And it's particularly effective if you have that phenomenon where clients start hammering services with retries once things get a bit slow or a bit flaky. And I guess throttling is a similar idea, where you put in a hard limit over time about how many requests can go from here to there, or a delay in between repeated requests.

For context, I was part of a team that was managing a very big installation of Konsole, which

has a bunch of really interesting design challenges. In certain circumstances, it's very prone to, say, thundering herds and to things really just started getting completely jammed up, is the only way to put it.

It doesn't have any way of timing out requests that have just been hanging around for a long time. So you can literally end up with your service just sort of sitting there doing nothing but managing a queue, which is not a great state to get into. But a few simple tweaks around that really, really helped the stability of those services.

CARLA GEISSER: I guess I'm in a little bit of a different place, because my whole role lately is to show up and provide the low-hanging fruit advice to whatever organization has hired us. And so that has been really fun. And I would say the most frequent advice that we end up giving to people is thinking about the priority of production operations against other things the business might be trying to do, or not.

A lot of people at this point have read the Google SRE book, and they know the words SLO and SLA, and they're very excited to start implementing those. But that isn't necessarily the place for them to start. In a lot of cases, they might have not quite enough production hygiene. They might not even know, really, anything about how their systems are behaving.

And so to start immediately with, oh, we need SLAs and SLOs for this product, doesn't help them. They just get mad about not knowing how their systems are performing. And so a lot of it is back to basics. What is your system doing? What is important about your system? What is the business function it is serving? Rather than any particular technical fix.

CODY SMITH: At Camus, the experience from SRE ended up being a core part of the product. We had to start with electrical utilities. Building trust. Really giving them a product that was lower risk.

Eventually, we want to get to grid orchestration, so we're actually controlling huge numbers of things on their grid. But before we do that, we have to prove that we can be predictable. And so we showed up at the office of our first customer and they took us into the control room and we got to see their systems. They had probably five or six different independently-deployed computer systems for different parts of their grid.

So a lot of the regular, day-to-day operations involved going from one system to the other, literally with things written on Post-its, back and forth. And so the initial deployment of our product ended up being like, let's take all this data from all these different systems and bring it together into one place. And so you have one web app where you can see most of what you need to know to run your grid day to day.

And then once we delivered that and they were happy with it, we were able to move on to the scarier product offerings, from their perspective. So monitoring ended up being a stepping stone

for us in our product roadmap.

MP ENGLISH: Sounds a bit like Mikey's experience, the Dickerson Pyramid that we hear about. I'm sure Carla hears about all the time. Yeah. Monitoring helps to start there, if you don't have it.

LAURA NOLAN: I wanted to put a little bit of an addendum on what Carla said there, about SLOs not being the place to start. That's something that I actually really strongly agree with. And think that there's been a real push that the first thing that you must do, if you implement SRE, is SLOs.

I think if you asked 100 people, probably 90 of them would say that, if they know anything about SRE. And think the reason that is is because SLOs are something that's standardizable. You can make a recipe for SROs and SLOs and you can build a product for it. And so you can say, do this. And the organization can at least start to do it.

But Carla's right. They're not necessarily going to do it well, because not every organization actually understands what is critical about their product. And not every organization actually has the right monitoring to have the right SLIs to really tell if the product is working or not. Like for example, think about services where data freshness is a big issue. That's much harder to measure than the typical examples of error rates and latency that people always give as the spherical web service example of SLOs.

So the work that Carla's consultancy does, it's not something that you can standardize in the same way. You can't build a product that's SRE in a box that will come and actually understand your systems and give you that context aware take on the-- I think there's something quite wrong about that. Always do the SLOs first [INAUDIBLE].

CARLA GEISSER: I think that's also evidence that whenever Google puts out a paper or a piece of documentation, people assume that it has been uniformly adopted inside of Google. And we all know from our own experience that that's not true. Even adoption of something like SLOs is very lumpy inside of Google. All of the various storage technologies that have been written up, adoption of those took over a decade in some cases.

And so that, I think, from the outside, is hard to balance with people, because they see it. And this is true with Facebook and any other company that publishes about their internal engineering culture. People assume that what has been published is how the entire company does it uniformly. And that will never be true. These things will always be deployed based on need or based on specific circumstances within a big company.

LAURA NOLAN: Absolutely. And you have plenty of teams, even in a big organization, that might have Potemkin village SLOs, where they have a thing that looks like SLOs, but it's not really that meaningful. They picked a random metric, and they picked a number that would be passing for most of the past quarter, and that's their SLO now, because somebody said that they

had to have SLOs. So even in big organizations, things are different between teams.

CODY SMITH: My job these days is to talk with, specifically in Google Cloud, customers about this kind of stuff. And I can completely agree with you that everyone wants to do SLOs first, even if they're not ready for them. I've had a couple of customers who simply just didn't have the metrics, like their observability didn't have the granularity or just didn't exist.

And they say, we want SLOs first. And it was like, well, that's going to be tough. I'm concerned that it's just that it's novel and it's slightly difficult to understand, but not very hard to understand. And so people latch onto it, because it's sexy and fun. But this also brings me back to the question from before, which is when you go into a culture or a team, do you go after the low-hanging fruit?

Like, we should stop doing the dumb thing over here. Or do you try to aim at something that has more impact and can be broadly applied across multiple teams or multiple services? I feel like that's kind of an art to decide on what to do, what's going to have the most impact on an organization.

LAURA NOLAN: You have to pick something that the organization is ready for and will accept and will see as useful, right? Not all of those things are going to be the same in every place. Some engineering teams will have had a bad experience with trying something in the past, and they will have a prejudice against it. So even if it might be the right thing to do technically, it might be the wrong thing to do organizationally, for example.

CODY SMITH: I would say anything I would have to add is kind of tangential to the framing of your question. But I would say the experience at a startup was counterintuitive to me in the sense that SLOs for us would really be a luxury. We are building product. We're building features. We're deploying our software.

We're integrating with existing utility systems. That's a ton of work. And the word "reliability" comes up fairly infrequently. I was expecting it to be much more front and center, because I know that utilities do care a lot about reliability for their customers. But mostly, what drives reliability that the customer sees are things like branches falling on wires and squirrels getting into transformers.

STEVE MCGHEE: You've all been in SRE long enough where you kind of get it pretty deeply. And now you've been exposed to other stuff out there and other teams. And I just wonder if you think that SRE has co-evolved anywhere else.

Or something similar to SRE, like is SRE really that special? And if you think it has replicas somewhere else in industries outside of ours, what does it look like? What qualities might those other folks share with us?

I guess what I'm getting at is, what actually makes SRE? Like, it's not just coming from Google. I think it was just out there in the universe already, and we just kind of stumbled on it. But I'm curious if you agree.

LAURA NOLAN: My grand unifying theory of SRE is that actually, SRE is just a production system specific manifestation of system thinking. And we do it in an informal way. For anyone who wants to hear more about this, I gave one of the keynotes at SRE [INAUDIBLE] about this just last month.

So what systems thinking is, it's basically the discipline of looking at how systems are structured in a sort of generic way. And looking at how those patterns and interactions in a system give rise to different behaviors. So I think for SREs, an example of a system's structure behavior thing that we're all familiar with is the idea of the cascading failure.

That kind of vicious cycle of load that causes retries that causes more load. And then you have to have an intervention to get out of that metastable state. And that's the systems insight. But there are lots more different kinds of systems insights, and there's lots of methodologies that we can use to analyze, particularly, the reliability of our system.

So there's a thing called East-BL, which is basically that something, something, something, broken links. And the idea here is you look at all the control structures in your system. So what's talking to what? And making decisions based on that interaction.

This could be your monitoring. Could be your health checking. Could be ORPCs doing automation between your systems. Could be interactions in your data plane. And you say, well, what happens if this link breaks?

What happens if the traffic increases by 100%? What happens if it's wrong? And that's a really valuable technique for looking at your system and saying, OK, well, how is the system vulnerable? How can I improve the design of this system to make it more robust to all these kinds of things that can go wrong?

If you look at SRE, a huge amount of system stuff is that kind of thing, looking at the system and understanding its behavior and how this part over here affects that part over there. And how the architecture of the system and the potential changes would make it better or worse.

So it's really about thinking about the system and how we can make interventions to make it better. And that's not just the technical systems. It's also the humans around the system. How can we be better informed? How can we work better together? How can we deal with on-call with less stress? All this kind of stuff.

So that's my unifying theory of SRE. And whether I think it's evolved elsewhere, yeah. I think there have always been great sysadmins who've done things very similar to the way SRE have

done it. Maybe not with some of the keywords like SLOs, but with a lot of the mindset. I don't think the mindset is distinctive to SRE.

CODY SMITH: I think there's relatively few systems thinkers in the world. SRE has a staggering fraction of them. And it's a really valuable high-level skill that maps well onto lots of other industries. I think the book I would recommend, if folks have not read it, by Donella Meadows called "Thinking in Systems," it's a really good primer on this topic.

CARLA GEISSER: For me, I guess the core thing about SRE is whatever happens when you take a small number of motivated and empowered operators and give them responsibility for something that is critical to the business, and it has to be responsibility for the entire chain of stuff. And so I think things that look like SRE or DevOps or whatever buzzword we're going to make up in five years, exist and have existed for a long time, just because of how you set up the people and how you set up their job. And as long as there's a small group of people who are empowered and motivated to own a critical system, then something that is like SRE will pop out the other end eventually.

MP ENGLISH: I think I heard two or three different major factors there, small group and empowered individuals. And maybe something along the lines of full stack scope. Why do you think it's those three factors that produce this generic SRE DevOps thing?

CARLA GEISSER: So I'll start with the end-to-end scope, because I think it's the one that's easiest to explain, maybe. And that is, there can never be another team who you can obviously point at and say, OK, cool. My responsibility ends here. It's definitely their fault. You have to feel a responsibility for the end-to-end system.

Even if there is maybe another team who you need to work with, your ownership needs to cross that boundary in some way. The small team thing is mostly about communications overhead and building a highly-effective group. I think there's a fair bit of organizational research that small teams just tend to work better.

And then they grow, because they have to, because the business grows. And then you split the teams apart along different boundaries. And this is like a normal cycle of how organizations behave. But for setting up a thing that looks like SRE, start small, for sure. So empowerment goes with the end-to-end.

Part of the problem is, they need to have the ability to actually change the thing that is in their way, or to demand change for the thing that is making the system unreliable. Because otherwise, you get back very quickly, and we have all been there, to the old school sysadmin world, where the software doesn't work. You have to deploy it anyway. And then everyone is yelling at each other all the time. And you just live in that world forever.

STEVE MCGHEE: When I was on shore leave from the spaceship, I learned the term "stay in

your lane," which I'd never heard before. Someone told me to stay in my lane when I was trying to help outside of my scope. And it cut me deep. I was not prepared to be told that. It was very strange.

And since then, I've tweeted about this a couple times. But telling an SRE to stay in their lane is completely antithetical to their entire being. It's like, I totally agree with that point.

CARLA GEISSER: And you could probably build a very good SRE team, just out of the people who have been told to stay in their lane at some point. That group of people, if you can find them, they would be a great SRE team.

STEVE MCGHEE: Yeah. They're hired.

LAURA NOLAN: This is one of the interesting challenges about SRE-ing in an organization that's heavily reliant on cloud services, because you do end up in a situation where you're trying to figure out how stuff is working and you're trying to figure out how to make things better. But suddenly, you have this big barrier in your way. And you're working with a service that you have very little visibility into, and go talk to the teams. And it really is quite a thing. I have not necessarily got a good answer to that.

CODY SMITH: I will say I think SRE prepares you pretty well for entrepreneurship. If you're going to found a company, it's a need to have a sort of sense of responsibility and ownership over the whole thing that makes you a good founder. So it's part of your every day, when you're in SRE, to feel ownership over the service that you're taking care of. And it puts you in the right mindset. And so I often encourage SREs to start their own companies.

LAURA NOLAN: Just to react to something that Carla said earlier, Carla said that small teams are a good thing. And also, that teams should have ownership over something. I think that's very right, because one of the anti-patterns that I've seen in a few places is SREs who try to act like permanent consultant teams, but without any actual ownership over any services.

And that's a weird place for people to be. I think there is definitely a role for short-term consultants providing that fresh view on your service. But when you have a team that, they kind of end up just project managing other teams to try and do self-service SRE, and the anti-pattern that I've seen is end up with a lot of very shallow checkbox processes that the centralized SRE team asks other teams to do.

So you'll end up with a PoR form and instead of doing-- at Google, if you were PoR-ing a new service, that's a production readiness review, you might expect to spend three to six months doing deep work, understanding that service, understanding its architecture and how it behaves and ways it might be improved, and instrumenting it and all of that good stuff. Maybe standardizing the tooling and how it does roll outs, all this kind of thing.

Whereas with the tick box PoR that I've seen, you just end up with people saying, yep, yep, yep, yep, yep. And there's little room for thought or deep engagement. It's just yep, I did the thing. I have some monitoring. Is it good monitoring? Who knows. But have some.

So therefore, I'm good. So you end up with this quite shallow engagement, and not the sort of deep context-aware engagement that I think is really what SREs should be doing.

CARLA GEISSER: I think that anti-pattern is particularly common in industries or organizations that already have a lot of compliance folks running around. So in the government or in finance industries or in health care industries, they already have a lot of people walking around with checklists to make sure they are complying with various things. And so it is easy for SRE or DevOps, or any of these kinds of specialized roles, to just become another version of that. Which, as Laura says, is not particularly useful.

MP ENGLISH: Yeah, there's different kinds of risk out there. And if you're already good at one type, it's really easy to try to pattern match the old risk model to the form of risk, whether it's reliability or liability, I guess. Didn't even mean to make that pun.

OK, so I think we have time for a couple more questions. Then we'll wrap up. Do you think it's more important to have individual SREs or to make reliability a part of every engineer's mentality? Like, everyone in the org.

So I guess it's, should you have these specialists? Is that the important part? Or do you want to distribute this speciality across your entire org? I guess you can also say both. But where would you start?

CODY SMITH: I don't think it's reasonable to have every engineer have a reliability mindset and understand the overall system, especially for more junior folks. It's just not practical to be able to do their core job and all of that other stuff. So the sweet spot is having, in a mature large organization, a few people sort of sprinkled in dev that think about reliability and understand it and apply it to the development process.

And then having a separate SRE team that really focuses on being able to meet the SLO as their primary objective. Those two work together, pursuing-- there's inherent tension between the two of them pursuing the goal of reliability and the goal of launching new features.

Those people on the dev side that are the reliability thinkers are golden, from my perspective. When I was in Search, this was Rob Stetz, who was amazing. He made the job of Search SRE so much easier. He was like the fifth Beatle. He was one of us in spirit.

CARLA GEISSER: I think, following along with what Cody said, the absolute worst thing you can do is have a huge number-- well, not a huge number. Even a handful of very competent SREs, and then sprinkle them throughout your organization sort of haphazardly. A lot of places

that I've consulted with try this, because someone tells them they need to do SRE, and then they go hire a dozen people with an SRE job title and put them in a variety of disconnected places throughout their organization. And it just doesn't work, because none of them can get traction and none of them can make any progress.

So I think if you're going to have a thing that you're calling an SRE team, it needs to be very dense with people who have the right mindset and the right level of authority within the organization. And then separately, I agree that having people scattered throughout the organization who are SRE-ish-- and in fact, in my career at Google, I sat on the official Dev side and on the SRE side probably four or five times, and bounced back and forth, because I never liked the idea that a line was there. So yes, having people on the feature-focused side of the world doing SRE things is also very useful.

LAURA NOLAN: And the worst passion is the single-embedded SRE that turns into the ops person and ends up being the person who's always called upon to do anything that's sort of production-oriented, like doing rollouts or doing config changes or managing Terraform and PromQL.

The problem with that is, you just end up doing all of the toil, all of the grunt work. And this is one of the things that I think Carla mentioned, that it's very hard to get traction in that role. I think that's because you essentially end up doing all of the housekeeping.

I think it's also very hard for career growth for SREs who end up in these embedded teams, because they have a different remit to the rest of their team. And so they don't necessarily have anybody working closely with them who can help them develop their skills and who can work with them on larger projects.

Very often, we see these career ladders where you're required to show that leadership and work with a bunch of people. And that can be quite difficult in those disconnected roles. So I think it can be very, very hard for people to get traction, both on their individual career and on being impactful.

STEVE MCGHEE: One final question. What do you think is standing in the way of the entire internet being more reliable? What is the problem? Is there a problem?

Imagine one. I'll give you 10 whole seconds to solve this problem. Why can't we just depend on everything on the internet working all the time?

LAURA NOLAN: This might be a controversial take. But by and large, the internet is pretty reliable, and so are most of the web services. Not all, but I think the story is actually pretty good. And the story is certainly a lot better than it was a number of years ago.

I don't have statistics on this, but I would strongly suspect that the biggest cause of disruption

for most people is probably their own local laptop internet. And that's not to say that we can rest on our laurels and say that, hey, software is good now because it's-- it's good, because a lot of people are working hard to make it reliable, I think, in most places. Even places where there aren't SREs, there are always a scattering of devs who are very reliability-focused in terms of the mindset.

So even if you haven't gotten a specific organizational carve out for SRE, there are people doing that work. The other thing as well is public cloud. And also, I guess, services like Bracell and Fly.io and all these kinds of fairly pre-baked kinds of environments. These things, I think, do help reliability, by and large. And so do things like Cloudflare or Fastly, right, that help you scale a lot of your static serving.

There's a lot of tooling there that's really, really, really helpful. That's not the full extent of the story of what you need, but I think it's an awful lot better than 20 years ago, when we were all running on a few pizza boxes in a rack somewhere, in a single location. The possibilities for what you can do reliability-wise these days are huge.

CODY SMITH: I would say, broadly speaking, validation is too difficult. And that's across the spectrum. I think at the unit testing end, the tools are pretty mature. And then integration testing, less mature. And then monitoring, probing, things like this, less mature.

There's just a lot of overhead to setting these tools up. And so relative to the amount of time you spend developing a new product or a new feature, getting good validation in place is pretty costly. And a lot of especially smaller companies don't have the time to build all of that stuff out.

If you're not a developer, if you're, say, an electrical utility and you have to put systems together from six different vendors or 10 different vendors to get a solar farm working, for instance, knowing that you've plugged all the pieces together correctly is very, very difficult. I would hope the next couple of decades reveal a lot more maturity in tools like that that give you the feedback right away that you've connected things correctly. That would be my dream for the future.

CARLA GEISSER: I kind of have a similar take to Laura's, which is like, things are pretty good. And also, to go back to Cody's story about working at a new phase startup, a lot of times, it's just not worth it to make it better. Because each additional line costs a huge amount of money. And for many products and many services, maybe that's fine.

So I think that's one way of looking at it. On a technical level, I think that the scariest thing about the internet at large is that a lot of it is written in memory-unsafe languages, which prove every year that there are security vulnerabilities or denial of service vulnerabilities, or just fragile software all over the place.

And that, I think, is the biggest risk to the internet at large. It's getting better in some areas, and

some companies are focusing on it. But we still have a lot of C and C++ in the core of the systems that keep the internet on.

LAURA NOLAN: So I was going to say, while I do think that the story about reliability is quite good, despite all of the hassle of wiring up all of these disconnected point solutions that Cody talks about, Carla is completely right in that the security, particularly around data security and confidentiality, I think the story is very, very bad. And that's somewhere where we're not seeing any systemic improvements. So completely concur with that. Maybe the future of SRE is security.

CODY SMITH: You can add an S to SRE. That should do it.

MP ENGLISH: Well, thank you all so much for joining us today. This has been a fantastic, insightful conversation. So thankful for all of you being here today. I have one last last thing, your parting thoughts. What is the number-one piece of advice you would give to teams facing reliability challenges?

LAURA NOLAN: I think the number-one piece of advice I would give to any team that is facing reliability challenges would be, don't just do the first thing that pops into your head to fix the problem. Because very often, that initial knee-jerk reaction is something that will maybe paper over the problem or kick the can down the road versus actually solving it.

Take that moment to take a breath and really analyze the problem and understand how it's working as a system. And understand how you can intervene to improve that. Maybe you do some quick-fix mitigations as well, but I think a lot of the teams with the worst reliability challenges are there because there's been many quarters of ignoring the problem, or sort of papering over the problem with quick hacks, rather than really doing the--

Maybe it's not a full redesign. Maybe it's as simple as a fairly small change. I'll give an example here. One time, I worked with a company that would occasionally do a thing where it would require all of the end users to update their client.

And sometimes, that caused problems. And the reason it caused problems was when a client would restart, it had to do a pretty expensive call back to base to get a bunch of data. And that call was rate limited.

And so sometimes, we'd tell all the clients to update. But then we'd rate limit them when they tried to restart. Now, it turns out that there was a rate limiter on the mechanism that required clients to update as well. But it wasn't synced up with the other rate limit.

So very simple systemic change was to align those two rate limits, and don't ever ask clients to update if they wouldn't get the permission to do the reboot [INAUDIBLE] call. And that was a pretty quick couple of lines change that made that operation safe. So rather than sitting there

trying to tweak those rate limits again, actually aligning the two rate limits fixed the problem, and fixed it for good.

So there's things like that. So avoid just doing what you've done before and kicking the can down the road, and really think deeply about your problems. That's my advice.

CODY SMITH: What I would say is, start with culture. Usually, there are plenty of people around. If problems are emerging steadily, it's because some number of the people around are not upholding reliability as a priority. They don't see it as part of their job.

They see it as maybe someone else's job, or just maybe a lower priority than launching a new feature. And if you can build a sense of personal attachment between every person on the team and reliability goals, whether they be SLOs or just fewer outages, it will take off on its own pretty quickly.

CARLA GEISSER: My thoughts on this are to start extremely small, smaller than you even think makes sense, with one user interaction or one piece of your system's flow. And work your way outward from there, because the thing I've seen most often in organizations is everyone is running around saying, oh, no. Things are unreliable.

But there's too many things. And so having a focus of, we're just going to make the home page load in under 500 milliseconds 99% of the time. That's our goal. We're not doing anything else right now. And then once you have that nailed, you've learned a bunch of stuff about how your system works and you can move on to the next one.

MP ENGLISH: Well, thank you all so much for joining us. I want to give a huge thanks to our guests, all former Google SREs. Carla Geisser of Layer Aleph. Laura Noland of Stanza, and Cody Smith of Camus Energy. I also want to extend a hug thanks to my co-host for this episode, Steve McGhee.

I also want to give a shout out to all of those behind the scenes that have made this season of The Prodcast possible. So huge thanks to Salim, Jordan, and Paul for all of the work they have done behind the scenes.

I also want to extend another thanks to all of my other guest co-hosts this season, Pam, Chris, and Rita. Thank you all so much for being here. Thank you all so much for your time. I am immensely thankful for everyone that contributed to making this season possible. Thanks so much.

STEVE MCGHEE: Thanks, everyone.

LAURA NOLAN: Thank you. It's been great to be here.

CODY SMITH: Thank you.

CARLA GEISSER: Thanks. Goodbye.

LAURA NOLAN: Bye-bye.

VOICEOVER: Prodcast, the Google SRE Production podcast is hosted by MP English and Steve McGhee, and produced by Salim Virji. The Prodcast is edited by Jordan Greenberg. Engineering by Paul Guglielmo and Jordan Greenberg. Javi Beltran composed the musical theme. Special thanks to Steve McGhee and Pamela Vong.

[THEME MUSIC]