Google Prodcast Season Three Episode Eleven

[JAVI BELTRAN, "TELEBOT"]

STEVE MCGHEE: Welcome to season three of The Prodcast, Google's podcast about site reliability engineering and production software. I'm your host, Steve McGhee. This season, we're going to focus on designing and building software in SRE. Our guests come from a variety of roles, both inside and outside of Google. Happy listening, and remember, hope is not a strategy.

—

STEVE MCGHEE: Hey, everyone. And welcome back to the Prodcast, Google's podcast about SRE and production software. I'm Steve McGhee, your host. This season, we're focusing on software engineering in SRE. This is notably done by humans, often in teams. Wow, I know. So bold.

These teams build and manage complex systems that tend to grow and grow. And what could possibly go wrong with that? Well, we have two great guests today with complementary approaches to big problems like this that we face in SRE, specifically, how do we deal with these big, complex, continuously changing systems?

How should these teams work together? How do they work together? And can one person even understand a big thing like this? And if not, then what? How do we even deal with this? So with that, let's meet our truly awesome guests, Christina Schulman and Dr. Laura Maguire, who will introduce themselves. Christina.

CHRISTINA SCHULMAN: I'm Christina Schulman. I am a software engineer focusing on reliability in Google Cloud.

LAURA MAGUIRE: Hey. My name is Laura Maguire. And I am a cognitive systems engineer, which is basically a fancy way of saying that I study how people do the thinking parts of their jobs.

And the cool thing about CSE is that we can study things like perception, attention, and reasoning capacities of people operating in all kinds of very cognitively demanding work settings. And we can apply those patterns to software engineering. So I like to say that software engineers have more in common with astronauts and fighter pilots than they might originally think.

STEVE MCGHEE: Cool. OK. So can we just start with this stuff is hard? These systems are big. We as

humans have trouble peering into each other's brains. So that also makes it a little bit difficult. So we do have to speak with words and write things down and plan for things.

So first off, when we're talking about these complex systems, I've also heard the word "complicated" thrown around and "it's big." What do these words even mean? What do we mean by this, complexity? Does it matter, the words that we use. And if so, in what ways should we be careful about how we describe these problems that we're trying to solve together?

CHRISTINA SCHULMAN: I think it's honestly difficult for humans to wrap their heads around just how large and complicated a lot of these platforms are. My mantra is that once a system won't fit on one whiteboard, you just don't understand anything that's outside the realm of that whiteboard.

The more whiteboards it takes up, the more you need people just to understand where everything is. And nothing ever gets less complicated. You just keep adding more and more complexity, because if you don't, your system will die and fall over.

LAURA MAGUIRE: Yes, to everything Christina said. And I would say that we tend to often think about complexity in terms of it being from a strictly technical sense. But complexity is a lot broader than that. And it extends to the whole sociotechnical system.

So I try to think about it in terms of the levels of abstraction. So what makes this work hard? What's cognitively demanding? How do we reason-- or how do we notice what is happening and what's changing in the world around us? How do we reason about that? How do we apply our knowledge?

And then when we start thinking about the teaming aspects, as Christina said, we need to start to bring other perspectives and other knowledge bases together. That brings in a whole lot more social complexity. And then organizational aspects, like, how are you managing trade-off decisions? How are resources being allocated? All of those things are stuff that software engineers deal with on a day-to-day basis. It's not just the technical parts.

CHRISTINA SCHULMAN: And then every time you add a new layer of abstraction to make things look simpler, you've just added an enormous new layer of technical complexity that your end users don't even know is there.

LAURA MAGUIRE: Yeah, absolutely. And you also start adding things like automation in. And now you're looking at human machine teaming. And how do we try to understand and manage with our automated counterparts?

CHRISTINA SCHULMAN: So basically, without putting some barriers in place, everybody winds up crying in a corner professionally.

STEVE MCGHEE: It hurts. I feel that so much part of our job as SREs is, of course, to try to, as we say, automate ourselves out of a job. Of course, we know that that's not actually true. The automation itself becomes a job, dealing with the automation and interacting with the automation.

One thing that we've found within Google-- having done SRE for a long time and having grown a lot, one thing that we have pointed out, I think, in one of the books is the idea of teams undergo what we call mitosis, which is when you-- the team is responsible for too much stuff. And so we've got to split the team in half along some line.

I know from experience, having done this several times, that figuring out what that line is is really hard and it's really important. And then coming up with the ability to judge who works on what and making sure that they actually understand what it is they're working on, super, super important, not technical at all. It's purely, as you said, Laura, sociotechnical. It is, "Who is doing what?" and, "How do they talk to each other?" and things like that. OK, that's awesome.

So another thing that comes up when it comes to understanding complex systems and the people that deal with them is stuff breaks. And one of the things we do in SRE is we deal with incidents. You can talk about being on call. You can talk about writing postmortems. You can talk about getting paged. You can talk about all these kinds of things.

What should we think about here when it comes to making sure we don't screw it all up? What are the risks, first of all, when it comes to dealing with these complex systems with a group of fallible humans that need to sleep and things? Where should we start with that? Let's start with Laura from the humanistic side. And then, Christina, I'm sure you'll have things to say, as well, on top of that.

LAURA MAGUIRE: So I think the point that Christina brought up about how there's a lot of different perspectives that are needed to really understand how the system works is thinking that we can try to solve these problems independently or that we can try to solve these problems without others and knowing when and how to bring other people in and how to bring them up to speed appropriately so that they can be useful to the incident response effort.

That in and of itself is actually quite a sophisticated skill set, especially when you're under a lot of time pressure, there's a lot of uncertainty. All of the details matter. And they matter more.

CHRISTINA SCHULMAN: Well, I'll just add that we actually have people inside Google who specialize in

dealing with very large, very visible incidents, even if they're not super large and visible, but who have experience in just doing the coordination and the communication necessary to keep an incident moving, while the people who actually understand the systems that are probably involved work on understanding what's going on and mitigating it.

I will say that, ideally, nobody wants an incident in the first place but you're going to have them. So being able to restrain the blast radius of an incident and hopefully limit the potential effects of any particular failure in advance is certainly something we would all like to do, although it's hard and it's very difficult to analyze, test, and guarantee.

STEVE MCGHEE: One thing that I heard some one of us was talking about-- before we started this podcast was the phrase that incident response is a team sport. So what you're both saying reminds me of that. And it helps to have, in your team, broad perspectives, whether it's just experience with the system at hand or if it's experience with the other teams that are involved.

So one thing that I've found when working with customers is lots of times teams that are adopting SRE or similar type of practices hear things like you build it, you own it. And they come to believe that means they should have complete control over their entire destiny. And that means they need to own everything from the load balancer down to the CPU for their particular service.

I don't think that scales. And I understand the idea of it is that you want to be able to have control over your own domain. But this doesn't really work when it gets really big. So does this ring any bells in terms of how to manage autonomy under scale? What are some ways of thinking about that?

LAURA MAGUIRE: Yeah, so I think from a cognitive systems engineering perspective, I think it's-- one of the fundamental truths is when you get to a certain size in a system of work, no one person's mental model about how that work system operates is going to be complete. It's going to be wrong in ways that can be consequential. It's going to be buggy in other ways. And so we do need to be able to bring multiple diverse perspectives to be able to respond to incidents.

CHRISTINA SCHULMAN: I would say that from a practical standpoint, you can't be responsible for too much of your stack. Among other reasons, somebody is going to need to switch out the software layers. And they aren't going to be able to do that if you are clinging hard to the behavior of your load balancer. There should be contracts around the behavior. There should not be any promises around how it actually operates.

I also think that in order to make it psychologically safe for people to be on call, it always has to be OK for them to pull in the people who do understand the systems that they think may be involved. And that's very

much a team culture issue.

You can shore that up with technical support. But there's no substitute for making it safe for people to get things wrong and blunder their way into being good at things. I do my best to get things wrong for my team as frequently as possible to model that kind of behavior.

LAURA MAGUIRE: I think that is a really important point. Because if all of our mental models are going to be partial and incomplete in some ways, then we're all going to be wrong at some point. And so being able to say out loud in an incident response, I don't understand what's happening right now, or here's what I know, because it's easier to say, here's what I know, and then here's what I don't know.

But normalizing that ability to not understand something and to be needing to ask or to help recognize when you might have something wrong, and then being able to share that knowledge across each other, that is fundamentally what's going to make your incident response work. I agree with you, that psychological safety and that social environment that allows you to just share knowledge.

CHRISTINA SCHULMAN: Makes it a lot easier to keep your rotation staffed as well.

LAURA MAGUIRE: And I think it reduces the anxiety of when you're going on call for the first time because the stakes aren't as high if you know that other people are willing to be pulled in, they're willing to tell you when they're wrong. And so it makes space for you to be wrong as well.

STEVE MCGHEE: Totally. So when there is an incident, I've seen within customers-- and personally, I've been on calls, not at Google but elsewhere, where you're in an incident like in the big room with the TVs on the wall and there's a VP in the room. And there is this idea of a chilling effect. And it is totally real.

It can be fine, depending on the VP and the culture and everything. But often I've heard and I've experienced that it's not fine. It actually tends to really make people freeze up and feel like they have to say the right thing. And they can't do what you're just describing, which is saying, I don't know. Let's look into it. I'm not really sure who can help, blah, blah blah, blah, blah.

So is there more to it than that? One of you also used the phrase earlier-- armchair SREing is another example of this type of thing. So can you talk a little bit about that? What is the effect of some other person in the room have on this team that's trying to get something done?

LAURA MAGUIRE: Well, I think it's useful to also think it can be a person in a position of leadership. And when there's authority and power imbalances in the room, then that makes sense. But it can also be someone that you really respect, that you don't want to be wrong in front of.

And I think that the difficulties-- or the folks who may have that chilling effect, as Christina was saying earlier, can be wrong, can normalize being wrong. But structuring your incident response so that the incident commander or the coordinator, whoever is in charge, can step that person out of the room respectfully if they need to and to be able to take a suggestion that comes from a person in a position of authority on par with a suggestion that comes from a junior engineer as well.

They don't have to give it more weight and more credence just because of who it comes from. I think there's an interesting practice within high reliability organizing, which came out of looking at operations on aircraft carriers.

And one of the principles of high reliability organizing is a deference to expertise. So just because the VP may have 20 years or 30 years of experience and they may have a perspective that may be a bit broader, your engineer who's only been on the team for two months might be closest to the action. And they might have the most current relevant knowledge to the situation. You dynamically shift where the focus is relative to who has the current expertise for that situation.

CHRISTINA SCHULMAN: I think it's worth bearing in mind, though, that if it's a sufficiently large and visible outage, the VP is probably-- they're freaking out, too, just as much as the junior engineers are. So having an incident commander who's in a position to manage the social aspects of both of those freak-outs is really useful. And to be fair, it's the VP who's going to get yelled at in public. The junior engineer is unlikely to get yelled at all.

LAURA MAGUIRE: I guess it depends on the organization for sure. But you're right, I think it's really important to recognize that everyone within the system has different kinds of pressures and constraints. And they are dealing with different kinds of goals and priorities.

When we talk about incident response being a team sport, that's actually a fundamental aspect of being able to signal to others when your goals and priorities are changing and when your pressures and constraints are changing so that they can proactively anticipate, or adapt, or adjust the things that they're suggesting or the way that they're carrying out their work to have this really smooth interaction and reduce some of the friction there.

STEVE MCGHEE: So speaking of teams and reacting to things, we also designed things as teams. And there's a law called Conway's law, which a lot of people might be familiar with. And maybe you can tell us a little bit about that, about how that might apply to the thing that you built and how it exposes itself to things like failure domains.

So what's going on here? And like what can we-- why is it worth knowing about this? What can we do about it? Christina, you're grinning like you have something to say. You've heard of this before, for sure.

CHRISTINA SCHULMAN: I mean, Conway's law is most frequently quoted, I think, in software companies that you ship your organizational structure, you ship your org chart, which I have mixed feelings about.

I don't want failure domains to look like my org chart. But at the same time, in terms of, as we keep saying, being able to understand the surface area of the things you're responsible for, there's really good reasons for that not to cross organizational boundaries. You can only understand so many things. You might as well understand the things that you're actually responsible for.

I think that you need very strong agreements in place in cases where the ways that you want your system to fail cuts across organizational boundaries. And I think it's a very hard problem.

STEVE MCGHEE: Can you give us an example of that, where you may have seen something similar to that?

CHRISTINA SCHULMAN: Well, I work in dependency management, so if you give me an opening, I'm always going to talk about isolating where your RPCs go. And then I'll keep talking about it until you hit me with a stick.

But fundamentally, we're running a whole lot of different things in lots of different geographical locations that are subject to different rules. Gmail is subject to different rules than, let's say, the various ML infrastructure pieces, both in terms of how they're allowed to fail and how they're allowed to store data.

When there's a failure, you want to constrain that geologically or geographically, if possible. If somebody trips over a power cord in South Carolina, you do not want that to affect jobs that are running in Europe.

But in order to guarantee that that's the case, you need to be able to understand and test behavior across a lot of different systems. And as we already said, since nobody can understand all of those systems in depth or in breadth really, you can't do that without software controls in place.

STEVE MCGHEE: The tripping over the cable reminds me of Leslie Lamport's description of a distributed system, which is any system where your computer can be affected by a computer you've never heard of. And I think that's great. That really describes a lot of SRE pretty deeply to me.

I've found actually that a lot of folks don't get this. There's a phrase that I think I learned from John Allspaw, but it comes from some other folks. And it's that of reductive tendencies. And this is just, “Give it to me simple. Quit with all the nerd stuff. Just tell me what's going on.”

And often we lose a lot of the really important subtleties when we reduce it too much. And this ruins the whole idea of what it is that we're trying to accomplish here by running a complex system.

If we write our rules for Gmail or whatever in these reductive ways, often we will lose track of some of the constraints that are really important that actually keep the whole thing running. Does this ring any bells either? This seems like it's crossing the two streams here a little bit.

LAURA MAGUIRE: Yeah, absolutely. This the work you're referring to is some folks from the Institute of Human Machine Cognition and from the Ohio State University. And they distilled, I think, 11 different tenets or truisms about managing or operating in complex systems.

And that synopsis you gave as like we want to oversimplify things because it's easier to manage and control and get our arms around of is actually quite dangerous because when we're treating things that are dynamic and they're simultaneous and they're things that are running in parallel as they're linear and they're cause and effect, this oversimplified models, we're solving the wrong problem.

Same things that a lot of these kinds of events and these failure events in complex systems, they involve a lot of-- they're nonlinear. They fail in surprising or unexpected ways. And so if you are not considering your system as being complex and adaptive, your response will not be complex and adaptive.

The law of requisite variety basically states that if your problems are all highly variable and very dynamic and changing, then your responses to those problems have to be similarly so.

So it goes back to what we said at the outset as like, how do we help people cope with complexity? And sometimes that's about helping to build common ground from the boundaries of where my part of the system and what I'm responsible for interact with neighboring parts of the system.

It's about helping people who have never worked together to very quickly try and ascertain who knows what, who's important, who do I need to coordinate and collaborate with? And those kinds of dynamic reconfiguration means you've all got to bring whatever skills and knowledge you have to bear in a situation that you've never seen before with people you maybe never have worked with before.

And so that's the ways in which we shift away from this oversimplified view of, we just write more rules. We just have good process. We can draw these really firm boundaries to say surprises are going to happen. How do we help people cope with that most effectively?

CHRISTINA SCHULMAN: I love that law of requisite variety. I have not heard of that before. I'm going to cite this now every time somebody asks why I can't come up with a simple design for a complex problem.

LAURA MAGUIRE: Ashby, 1956.

CHRISTINA SCHULMAN: All right. I'm going to read this now. I will say that most of the really interesting large outages that I have seen in Google production infrastructure have involved interactions between systems that were very powerful and very complex. And they had to be that complex in order to handle just the enormity and the heterogeneity of the system.

And there's a point at which you simply can't prevent these interactions from happening. You can't predict them. You can't prevent them. You just have to have really good systems in place for containing and mitigating them. At some point, I don't care what the root cause was. I just want to be able to make it stop without spending three weeks understanding it.

STEVE MCGHEE: Yeah, stop the bleeding is a term that we tend to throw around. Sure, we'll find what really happened someday. But for now let's mitigate. And let's bring the ball forward.

LAURA MAGUIRE: But it's interesting, too, because sometimes the right response is to let the bleeding continue a little bit more so that you have more diagnostic information. So this goes to those like, how do we manage these trade-offs? How do we make sure that that VP is in the room so that their perspective can be included in what types of actions we're trying to take?

CHRISTINA SCHULMAN: That's why we love observability.

STEVE MCGHEE: I know we just talked about how reducing a system to a simple idea is actually not great. But there is a metaphor, if you will, that comes up quite a bit when we're talking about these types of things. And that's the idea of aerodynamic stability, which comes from flight, from airplanes.

And this is the idea that you want a system that when you take your hands off the yoke, it satisfies itself. It brings itself to some level of flying ability. I think this was coined by John Reese within Google, at least. I'm sure it's been used elsewhere a million times.

But how can we use this metaphor when it comes to designing systems that don't require an intensive hands-on keyboard at all times and like watching all the screens? Is this a real thing? Is this something that you can actually strive for? Or is this just a pipe dream from JTR?

CHRISTINA SCHULMAN: JTR, John Reese, was specifically talking about removing dependency cycles from production when he wrote about that. And essentially, what you want to be able to do is identify and remove dependency cycles from your system.

A dependency cycle is when you have system A depends on system B, system B depends on system C, system C depends on system A. This seems like it should be easy to identify and prevent. But when there are 300 systems involved in this cycle, it's a lot harder to find.

And it's particularly pernicious when you have turn-up cycles, like, everything went down because somebody tripped over a very large power cord. Bringing it back up is not going to be able to happen automatically if you have these cycles.

The problem is, of course, that we keep saying this is a very large, complex system. So massively re-engineering things to change across multiple systems. It can be done. It hurts like hell. So putting systems in place where you control how things are allowed to depend on other things. Essentially the top of your stack is allowed to depend on the bottom of your stack. The bottom of your stack should not have dependencies on the top of your stack.

That's going to make things a lot less painful when something at the bottom of your stack goes down, comes back up, and you want everything at the top of your stack to recover without human interference.

STEVE MCGHEE: Yeah, just being able to speak in terms of directionality and having a common understanding of this is what we mean by up and this is what we mean by down and there is this intent to not have this particular direction, that helps a lot.

LAURA MAGUIRE: Yeah, this may not be directly answering the question. But it kind of brought up a really interesting thing. I saw in some studies that I did watching ongoing operations within large-scale systems. And that is that people are nearly continuously monitoring the systems and providing small little course corrections. And they are preventing incidents before they even-- before they're even a seed of an incident.

And so this is something that's subtle but very nontrivial because a lot of this work is hidden. All of the ways in which someone's sitting in a co-located space and they spot something on a dashboard and they go and check some logs or they do some little action to make a change in the system there, this stuff is happening all the time, all around us. And we don't typically tend to notice it.

And so I'm a paraglider when I think about dynamic flight under my wing. It is these really small little course corrections because you don't want to drive things into the place where it is very clear that you need to make a correction.

So the point, I guess, of bringing this up is that this hidden work, this continuous monitoring and continuous managing of the system is work. And if we overload engineers with task work, or support work, or feature

development, or whatever it is, you're going to lose a lot of this capacity that's already happening.

And so maintaining a little bit of slack in the system or starting to notice when and how people are doing these small, little course corrections can help to actually surface them so that you can account for them, you can resource them, and you can try to engineer them into future systems to prevent surprises.

CHRISTINA SCHULMAN: I'm working on a project right now that is very much affected by-- we're trying to develop some guarantees around a system that involves a lot of that constant attention and babysitting.

And it's very dangerous because you really want to-- you want to design for a system that doesn't need that. It can be such a hidden thing. And if you assume it isn't there, you're going to have a very exciting time.

LAURA MAGUIRE: Yeah. And I would posit that from studies of any kind of process control situations, that this is a fundamental principle of these systems. And while we don't want it, we might actually need it to account for the things that we can't imagine in our design or account for the ways in which interactions or pressures from the outside world are influencing the system in ways that we don't expect or anticipate.

And so I agree, we don't want it in the system. But it's also true that it's there. And so being able to keep that additional float, that additional capacity to maintain it, remove it where we can, but don't eliminate that entirely, I think is very necessary.

CHRISTINA SCHULMAN: I think you're probably right. But, boy, I hope you're wrong.

STEVE MCGHEE: There's a good story that I may have shared already on this podcast, but I'll go for it anyway, which was, there was an SRE who was in charge of a thing near the very front of all of Google, near the load balancer.

And he clicked a button and pushed a thing. And all of a sudden, everyone around him, everyone's pager went off at the same time. And he went, whoa, hang on a second. Is this blah, blah, blah? And they're like, yep. And he's like, hang on, click, click, click. OK, I fixed it.

And so it was a significant dip in the all of Google graph. And so I tell this story to customers all the time. And I say, OK, if you're the manager of this person, what do you do? Let's talk about incentives. Do you take him aside and have a quiet word and say, we'll never speak of this again?

Do you fire him on the spot? Do you dock his pay by two weeks? What are all these bad things you could do? And what happened at Google? What happened with an SRE that was actually really good?

They put him on stage. And they had him tell his story to the entire company. And they showed this stuff

happens. And his reaction was very, very good, the fact that he was able to notice it, adapt to it, fix it, and be like, OK, it's good. And also, I added this preventative step so that I don't screw this up again in the future.

Instead of just hiding the problem or expecting people to never have this problem, they declared this problem to the entire company. And everyone went, oh, OK. Not only can this happen and I should think about this type of problem, but if this were to happen, I wouldn't be fired. I might get put on stage. This could actually be good.

So a lot of these structural things when it comes to incentives and-- we talk about executives doing terrible things, but they also do good things, too. Leadership can definitely make the right culture as well.

So this is just another way of looking at this problem as well, is, like, when there's this complex stuff and these emergent failures happen, how you react to them is often a byproduct of just how you interact with everybody else within your team over time and how you observe that happen with other people as well.

LAURA MAGUIRE: Yeah. Well, I think it's really important, that organizational response to failure, because if they were to dock his pay or take punitive action against him, that drives a lot of the reporting underground. It drives a lot of the conversations of, oh, I thought it worked this way. And then I found out in a very big, public, embarrassing way that it didn't.

And so the more that these things can come to the surface, these flaws in our mental models and we can talk about them and we can share that knowledge, the more resilient your system is actually going to be. So I think that was a great response.

CHRISTINA SCHULMAN: I think the most famous intern incident in Google's Pittsburgh office was when one of the interns-- and this happened years ago now. But one of the interns on the payments team was validating credit card processing.

And on her last day, the whole team had taken her out to lunch when everybody's pagers went off because her test had broken payments, all the payments. And she was, of course, horrified. And the team was delighted because her test had successfully found a problem. She did wind up converting. And I think she's a manager now.

But we tell that story to all the incoming interns that they understand, A, you're going to be working on real production systems and you're going to have a lot of power. And B, it's OK if you have a highly impactful event.

LAURA MAGUIRE: Yeah, I think Etsy has the three-armed sweater that they present to one of the biggest, most impressive failures in prod. And it's kind of a nice way of normalizing that failure and of surfacing some of these things.

STEVE MCGHEE: Awesome, yeah, that's important. Well, thanks to you both. I don't want to keep you here all day. And I know we could talk about this all day, so we're going to have to cut it at some point. But before we go, if there's anything that you want to share with our audience where people can hear more from you on the internet or just like a pithy truism that you would like to share with the whole internet, now's your chance.

CHRISTINA SCHULMAN: I would just like the whole internet to check their return values.

STEVE MCGHEE: Excellent. Good advice.

LAURA MAGUIRE: Connecting with me on the internet, I guess I can be reached by email at laura@tracecognitive.com, or I'm on Bluesky, lauramaguire.bsky.social. You can find me at the Cognitive Systems Engineering Lab at the Ohio State University. And the takeaway, I guess, would be don't oversimplify. Lean into the complexity. And you're going to have a lot more adaptive and resilient responses.

STEVE MCGHEE: Awesome. Thanks to you both. And thanks to all our listeners. And as always, may the queries flow and the pages be silent. So long.

LAURA MAGUIRE: Bye.

—

[JAVI BELTRAN, "TELEBOT"]

JORDAN GREENBERG: You've been listening to Podcast, Google's podcast on site reliability engineering.

Visit us on the web at sre.google, where you can find papers, workshops, videos, and more about SRE. This season's host is Steve McGhee with contributions from Jordan Greenberg and Florian Rathgeber. The podcast is produced by Paul Guglielmino, Sunny Hsiao, and Salim Virji.

The podcast theme is “Telebot” by Javi Beltran.

Special thanks to MP English and Jenn Petoff.