Google Prodcast Season Three Episode Four

[JAVI BELTRAN, "TELEBOT"] STEVE MCGHEE: Welcome to season three of the podcast.

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.

JORDAN GREENBERG: Welcome to The Prodcast, the Google SRE production podcast.

My name is Jordan Greenberg.

I'm a PGM in GCP security.

And with me co-hosting, we have--

STEVE MCGHEE: Steve McGhee.

I'm a reliability advocate for Google Cloud.

And our guest today-- LIZ FONG-JONES: Hi, I'm Liz Fong-Jones.

I'm a former Google SRE, and now I'm the field CTO at Honeycomb.

JORDAN GREENBERG: How about it?

I love a little bit of popcorn to start the day.

By that, I mean it's late at night, so popcorn is extremely appropriate.

But what would be popcorn without a little bit of entertainment, which is why we have here today, Liz, to tell us all about SRE stuff, which, we could argue, is some of the more entertaining stuff out there.

STEVE MCGHEE: Liz, you work at a company, I believe, called Honeycomb, is that right?

LIZ FONG-JONES: Yeah, that's correct.

STEVE MCGHEE: And today, we are hoping to talk about observability.

Would you say that's something that y'all are interested in and are professional speakers of?

LIZ FONG-JONES: We're definitely more than interested in observability.

We're definitely the ones who popularized it to the point that people have just decided to rename APM to observability, which is similar, in a lot of ways, to the challenges of people just saying, oh, we're just going to rename our sysadmins to DevOps or SRE.

So that's been a fun landscape.

But yes, we popularized the word "observability," at least when it comes to software development.

STEVE MCGHEE: Cool.

Can you give us a couple quick definitions?

And do you think they're stable around APM, monitoring, and observability?

Because I know as an old person, I've been doing monitoring for a long time, but the new cool kids, they don't do monitoring, they do observability.

And they tell me that I'm wrong and old.

That doesn't seem correct.

I feel like I'm still right.

What's going on here, Liz?

LIZ FONG-JONES: I think everyone has a perspective that's informed by the problems that they've been solving, right?

So when you think about the reasons that monitoring came into play, they came into play because we wanted to understand when our systems were broken.

It came into play when we wanted to have charts and metrics to try to understand what's going on.

But as our systems got more complex, it no longer became possible to, let's just attach a local debugger, or let's just scroll through the logs.

And now, we start having distributed systems where you really need a deeper degree of introspection.

And that's kind of where we borrowed the word "observability" from control theory, to talk about of now these systems that are black boxes-- how do you make sure those black boxes are egressing enough telemetry for you to be able to reason about what's going on?

So, actually, it's been funny-- over the past year or so, my boss, Charity Majors, and I have basically thrown up the white flag for it-- we concede, fine, if you want to call monitoring observability, that's cool, right?

It is a degree of observability, it's just not necessarily the highest degree of observability.

So we've started thinking about observability as a spectrum, right?

You can have no observability, like you can't see anything that's going on.

You can have limited observability, or you can have more complete observability coverage.

So that's kind of where we start talking about this observability 1.0 versus 2.0 thing, right?

In the previous world, we used to have logs and metrics, and that was sufficient to debug our systems.

And now, we're starting to use on-demand analysis of data, whether it be tracing data, whether it be structured log data, to derive real time insights and to look at end known end known problems rather than just the things we thought to monitor in advance.

STEVE MCGHEE: Yeah, that makes sense.

JORDAN GREENBERG: So, you're saying some things that I, as a person who am on the tertiary side of SRE, tangentially, I see into that.

You were talking about how it's sort of an observability black box.

LIZ FONG-JONES: Yeah.

JORDAN GREENBERG: How can we really know how the system is working in that way, what is seen as the x-ray machine, to get into whether or not something is performing as it's supposed

to, whether or not it's reliable?

LIZ FONG-JONES: Yeah, so I think that definitely there is some analogy to x-rays or to looking at the monitors on the black box.

But I think, really, the folks like John Allspaw who have been kind of thinking about things in terms of a systems perspective, they really have it right in terms of it doesn't matter what this image or this combination of white patches and dark patches-- that doesn't necessarily mean anything until a person interprets it.

So the way that we should be thinking about this is you have in your head a mental model of how do you think the system is functioning-- how do you test that against reality, right?

How do you reconcile that with what's actually happening inside the machine?

This is where the scientific method comes into play, right, where you ask a question that is falsifiable.

If you can refine your understanding about what's going on, whether it's working, whether it's not working, whether it's misbehaving slightly-- to understand why what's going on, you can't just necessarily directly look at the image of the blob of pixels and be like that's the problem, right?

Sometimes, you have to decide where to shine that ultrasound probe, right, or where to run that X-ray, and then to interpret the results, and then refine your model, and iterate until you understand.

So I don't think it's a one and done thing, right?

I don't think it's a you go to the doctor, and they just do the X-ray, and you're done, right?

It's more of an iterative process because our systems are so complex these days that you cannot diagnose them just with that single image.

STEVE MCGHEE: My favorite thing about that analogy is that's not even how doctors work.

If you go and get an x-ray, there's way more stuff that they talk about.

It's not just the one x-ray that they talk through with lots of stuff.

Sure, if one major bone is broken, maybe that's it-- if it's simple.

But when it's more complex, yeah, they got to do lots and lots and lots of tests and things like

that. LIZ FONG-JONES: Yeah, exactly.

They have to do lots of tests to refine, and hone in, and exclude red herrings.

And the other thing that I find really fascinating-- this is a complete tangent, by the way-- JORDAN GREENBERG: Love it.

LIZ FONG-JONES: I was watching a video about why it is that you can now take, at least in the UK, you can take liquids through security.

And it turns out that there's all this complex software that goes into analyzing the results of the x-rays.

And there's multiple beams at multiple frequencies and they're rotating around your payload, right?

Like this is why they call them the CT scanners and not just x-ray scanners, because they're actually generating a 3D image, and they're kind of computing differential density.

That's why they can say, oh, this is a liquid.

This is definitely water versus something else.

This is something that looks organic.

This is something that looks metallic.

It's not just a, is it dark or is it light?

There's actually a lot of really complex math that goes into this to be able to render results that a human being, who's not necessarily the highest paid person in the world, can sit there and interpret it and understand.

That's kind of what my mission is as someone who works on observability, right?

How do we get all of these complex signals and systems to dance together so that we can make it as easy as writing a SQL statement for someone to figure out what's going on with their systems?

Or, even better, what if you didn't need to write the SQL statement?

What if we just highlighted what was interesting to you?

So that's what I think is the kind of challenge ahead of us and why I say, wrapping it back around, observability is spectrum, right?

If you have basic monitors, that's a low degree of observability.

It's better than nothing, but there are so many better things that you can do that will allow you to figure out what's going on faster in your systems if you adopt newer technology.

STEVE MCGHEE: Yeah.

Another thing to point out with that analogy, I think that's a great way to think about it is at one point, that technology to look at liquids changed.

It got better.

They released the next version of the thing.

And policy shifts because of that, because we actually have a better understanding of, in this case, the payload, the baggage.

And so we can actually change the greater system because of a piece of technology improving.

And the other half of this is the people who are in charge of the policy might not be even aware that this piece of technology changed.

And so being able to have components of systems be, quote, "upgraded," and then having an effect ripple through the rest of the system because of that, whether it's good or bad or whether it's just changing the way that you do interpretation of the system, like you said, your mental model of the thing might not match reality anymore because someone pushed a change last week where we don't use that thing anymore that you remember it used.

So I think that's a great way to think through it all.

Yeah. LIZ FONG-JONES: Yeah.

This is why Charity Majors and I have this kind of rallying cry of, you should be able to deploy on Fridays, right?

When I was a wee SRE you would yell at someone who deployed on Fridays, right, because it was like, oh my god, what if it breaks?

What if it has a delayed reaction?

What if it goes wrong in production?

JORDAN GREENBERG: Yeah.

LIZ FONG-JONES: But I think that was born of a lack of observability, of a lack of confidence in production, that, ideally, if your systems are well instrumented, you can push something and watch it for two hours afterwards using the best available sensors and technology.

And if it hasn't blown up after two hours, right, you can go home because it might blow up three days from now, it might blow up three months from now-- but either way, does it matter whether it was pushed on a Thursday or on a Friday if it has a delayed effect of five to seven day?

So I think this is really the advantage that having better confidence in production gives us is it gives us more confidence to ship software more quickly.

We as SREs overfocus sometimes on the reliability part, right?

And the reality is we are a service function, right?

We are here to serve the users and to serve the developers in being able to get features out to production in a reliable manner, not to just gatekeep and say, no.

There's the apocryphal story-- it's actually real, but it's been played up a little bit-- about this at Google named Bogdan, right?

And there is a user device added to the Google kernels that was if you cat proc Bogdan, it would say "no." And I like to think that we've moved away from that kind of bastard operator from hell kind of mentality of, oh, it's the users that are wrong.

We say "no" to everything.

JORDAN GREENBERG: OK, so that is a good question.

It does lead into when you are considering whether or not to deploy something on a Friday before a new video game comes out that you desperately want to play in a few hours, how do you think about deploying it?

What do you think about deploying?

And what do you think about when you're implementing a new fix, new piece of technology, new software, to mitigate as much of the, oh, no, we have to stay online super late to try to fix

this situation?

Because we all know about the post mortem, which happens after.

But what comes before a launch comes out of an infrastructure or software change that is the thing that makes it so that nobody has to write the post mortem.

LIZ FONG-JONES: I love the pre-mortem.

I think it's great, right?

You comprehensively think about, what are all of the possible things that could go wrong, understanding that you will miss things that you couldn't have foreseen.

JORDAN GREENBERG: Right.

LIZ FONG-JONES: But identifying in advance what those risks are and taking measures to mitigate them, or at least to know if you're heading into dangerous territory, that's kind of the first piece, right?

So when we think about common mitigations, we think a lot about canary deployments, we think about rollbacks-- although, funnily enough, it turns out that your rollback can also further damage your system if it's something that's caused by change.

But, obviously, there are going to be unknown unknowns, but kind of making sure you're on top of your known unknowns is kind of the first thing.

And then the other thing is making sure that you have adequate instrumentation, that you have adequate telemetry to know when to pause doing that rollout, when to pause the thing that you're working on, because, if things continue in that direction, it's going to be bad, right?

So I think that's how you can take a lot of that risk out of the equation by having the visibility, right, to see, am I steering the ship in the wrong direction?

Of course the power could go off on the ship and you could steer directly into the bridge.

But you don't want to deliberately steer your ship towards the bridge or to have no idea-- if your radar is not functioning, you shouldn't be driving the ship, right?

I think that's probably the best way to think about this is, how do we maximize our controllability and our observability to go back to the control theory stuff, right?

How do we make sure that we can actually figure out what's going on and make sure that we

are taking small steps to go in the right direction at all times?

JORDAN GREENBERG: OK, awesome.

STEVE MCGHEE: So one thing that's pretty, at least well talked about, and depending on where you are, it's pretty well understood is the idea one of the main things that you want in observability is less about the guts of the system and more about, are users happy?

We talk about SLOs a lot.

I think there's other ways to measure this.

LIZ FONG-JONES: Yeah.

STEVE MCGHEE: How do you come across this?

Or do you start with that?

Do you find that it's a good idea to start with, how do we define what it means to make users happy?

And then how do we know that's actually happening?

Is that a valid place to start?

LIZ FONG-JONES: I think it's a valid place to start.

And, actually, one of the reasons why I went to Honeycomb after Google was I saw so many people struggling with the "we've done SLOs, now what?" Now, we've got this additional metric that is very vague that just says, something is wrong.

And if you don't give people tools to investigate something is wrong or users are not happy, then you're not actually adding value to people's lives, because all you're giving them is a dashboard that turns red or a quarterly report that they have to produce that says, we missed our SLO. And it's like, great, now what?

So I think the most powerful thing that we can do is to make sure that our SLOs are driven by data that is well-observed so that if you have a signal of user transactions are exceeding 3,000 milliseconds, that user transaction had better be part of a distributed trace or a structured log so that you can follow it all the way through so you can bisect the problem and figure out, where is this coming from?

Is it from this or from that?

Which population of users are impacted?

That way, it becomes more of an interactive tool, right, where you can steer, where you can say, you know what?

My SLO's starting to burn, and it's coming from that component, I'm going to freeze that component, right?

Rather than, oops, we burned through the SLOs, and we have no idea why, right?

Or our SLO is alerting, and we can't figure out what's going on.

So I think that SLOs are really the ultimate, of course, symptoms of user pain monitor.

But you have to be able to correlate that down to figure out which potential cause is implicated.

And people have gotten the cart before the horse, right, in the past and been like, OK, like we're going to put AI ops, and we're going to use AI to sift through our potential cause based alerts.

And it's like, no, you have the data there if you're doing tracing.

Wire your SLOs to that ability to drill down and investigate.

So that's why I came to Honeycomb and I was like, oh, my god, we have the perfect ability to investigate things, and we don't have a SLO product yet?

OK, let's wire up SLOs to Honeycomb traces and, bam, now it's one of our top selling features.

It's great.

JORDAN GREENBERG: So you kind of got into the next part of the question that we were going to ask you.

What is the downside, or, maybe, is it controversial to say, some people would think that the more you can pull people out of system-style monitoring, the better it is, because then there's no human error of, oh, we didn't see this or this is so much better because the robot found it.

We trust the robots.

Do you agree with that statement?

Do you want to talk about that?

LIZ FONG-JONES: Our job as a SREs is to create systems that are safe.

If we do not have exposure to those systems, we're going to start drifting away from knowledge of what those limits of those systems are.

So I think that, yes, if something is toil, and it always works in the same way, and the robots can fix it for us, great, right?

Or if your neighborhood friendly cloud provider can abstract it away from you, great.

I don't have to think about whether my SQL database failovers are working, right?

That's a problem that the Google Cloud SQL SRE team deals with.

So I think there are areas where you have to be like, OK, I'm going to bubble that abstraction, right?

Either a robot is handling that or another team is handling that.

But for your own systems, I think the role of machine assistance is to help us as a friendly assistant and as a copilot, but not necessarily as the primary driver.

So the classic way that I think about this is, for example, two of the technologies we built into Honeycomb, right?

One technology is if you draw a box around an anomaly, we're going to tell you, these are the things that might have changed in that anomaly, right?

These are the things that are correlated with that spike in latency.

And maybe it's causation, maybe it's correlation, the AI doesn't really know, but it does say, here are some follow up questions you might want to ask to refine your model of what's going on with the system.

Remember when we were talking about that at the beginning, like, the job of observability is to help your human brain reconcile what's going on in the system?

If you're outsourcing that understanding the system to a machine, then you, number one, you're now steering the system without having any idea of what's going to go wrong with it.

And that direction lies 737 MAX situations where people think the system is doing one thing and they don't actually understand the system, because the models have diverged.

So it's really important to guide human beings towards asking the right questions so that they refine their knowledge, right?

This is the same reason why we don't say, oh, my goodness, with Coursera, you can have a robot learn the thing for you, right? No.

Coursera is a robot that teaches you things, right?

What's the name of that company?

Khan Academy.

Khan Academy is not a robot.

Khan Academy cannot learn things for you.

Khan Academy can use AI to determine what to teach you.

So, yeah, I think refining your model of the system so that you can better and more safely operate the system.

So Honeycomb's bubble up feature, suggesting what questions you should ask-- where should you look?

But it doesn't actually say, this was caused by that, because that's the only determination you can make.

I think the other thing is helping people formulate their question breezily, right?

People don't have to learn Honeycomb's query language anymore.

They can ask a natural language question, and it'll automatically get translated into Honeycomb's query language.

And over time, people will learn a query language by kind of having that catalog of examples of, I typed in this natural language question, here's what it output.

OK, now, next time, it would be faster and more precise for me to enter the query by hand, right?

So that's the spectrum of how I think about AI assistants in the field of SRE.

Of course, I'm sure that you have had or will have Todd Underwood on the prodcast in the future, right?

I know Todd has some very fascinating thoughts about this topic as well.

STEVE MCGHEE: He has opinions on everything.

I'm pretty sure of that.

One thing that came up recently on the interwebs-- you and I had a very mild disagreement that I thought it was kind of fun.

It wasn't like that.

Don't worry, Jordan.

Specifically, I posted a thing that I suggest to some new teams, which is don't go looking for problems in production.

Don't go trying to find problems if you don't have a reason to.

And your response to that was really good, I thought.

It was kind of like, yeah, you should, actually.

There's good reasons to be poking around.

I was hoping we could clarify that a little bit today.

LIZ FONG-JONES: Yeah.

STEVE MCGHEE: What did you think was going on there?

LIZ FONG-JONES: So I think that it's really important for people to have a baseline as to what's going on in their systems day to day, because if not, you wind up having that looking for problems moment when there's an actual incident going on, and you cannot tell whether you are fixing a real problem or whether you are fixing a less important problem.

STEVE MCGHEE: Totally.

LIZ FONG-JONES: So I think kind of going off of what I said about refining your mental models,

if you never investigate production, you're not going to have a great mental model of how production works.

JORDAN GREENBERG: Right.

LIZ FONG-JONES: That being said, if you're prioritizing doing a super important feature versus shaving off 20 milliseconds of latency for 1% of users just because it seemed, hey, that looked weird, prioritize doing the feature, probably.

But I think I've talked about this before in other mediums-- if you're on-call, if you're dedicated to on-call this week, and you're not meant to be doing any feature tickets, spend your time looking at production, both to get a sense of what do things look like this week.

And also, your job is not necessarily to be working on tickets that week.

You may as well go and file down any rough edges that you find while you're poking around, right?

So I think there's that balance of dedicate time to looking around in production, but don't let it distract you, right?

Don't let perfectionism detract from getting your job done.

STEVE MCGHEE: The other side of, just to defend my position, not that that's important, is there is also the idea, though, that distributed systems can tolerate some forms of failure.

So if a component is not great in some way, but the system is fine, the system is fine.

So being aware of that is super important.

It's super great.

Doesn't mean you need to go and fix that one thing or demand that that one thing never be lossy again, because as long as the system is capable of managing around that-- you may even be in a degraded state.

That's really good to be aware of because you don't want yet another thing to fail on top of that.

LIZ FONG-JONES: Exactly. STEVE MCGHEE: Yeah.

LIZ FONG-JONES: You have to know, what is your safety margin?

Am I in a degraded state or not?

If you think you have extra layers of defense that you don't, that is a problem, right?

You will make bad judgments.

JORDAN GREENBERG: So smart.

So this is actually applicable to a lot of things, not just SRE-- understanding what a baseline is in general and understanding what can give is a concept that can be applicable to everything.

And, as you know, I'm PGM, so I'm going to be wearing that hat just a tiny bit.

Consider this is the same way that we deem what priority is in terms of what feature is next.

It's the same thing as what thing needs to be fixed in triage for what's on fire.

It's kind of the same concept in general.

So knowing what your baseline of, oh, this is what prod looks like-- I know that things are mildly meh here, that helps me define in the next feature, I can make this better because I know what that looks like.

But if you don't know what that looks like and you don't take the time to get there and figure that out, you might not know that a feature is needed or a change is needed to improve that space for yourself, or users, or end users who are like, well, this is a little bit laggy for me, but I'm not going to complain because it does work.

LIZ FONG-JONES: Yeah.

You kind of want to catch things before they catastrophically break, right? JORDAN GREENBERG: Exactly.

Exactly.

There could be symptoms of, oh, I'm feeling a little bit sick, but I'm still fine.

STEVE MCGHEE: Yeah.

So, first off, how do you, Liz, see SRE changing in the next two years?

Do you see it changing in the next two years, and so not just for your team, but the industry?

LIZ FONG-JONES: I think the job title SRE is going to continue to exist.

I think the idea of having a department of SREs is no longer necessarily fit for purpose.

So we are seeing this migration of teams towards the platform engineering model where you are uniting SREs and people who are doing build systems, people who are doing UX platforms and scaffolding for building UIs-- all of these engineering productivity functions are coming under one roof.

And I think that's a really good thing because at the end of the day, we are all serving different facets of the developer experience and the end user experience, right-- security being another one of them, right?

So when you think about it that way, it makes sense for us to work together with other teams, like security, like build pipelines, to make sure that experience is seamless and that we're able to force multiply and help software developers do the best jobs that they can.

So I don't think we should cling on to there will be directors of SRE and VPs of SRE anymore, and instead kind of really embrace the idea of, what does it look like to have this more kind of holistic view of developer productivity with SRE as part of it?

STEVE MCGHEE: Thank you.

JORDAN GREENBERG: OK.

Building off of that, there are people who are maybe in college right now who are thinking about going into some sort of software something-- maybe it is SRE, maybe it's something else.

Going even further than that, what kind of advice would you give to today's high schoolers about, those who are interested in making the internet better, bigger, faster, stronger, for all of the use cases that we as people want to use it for?

LIZ FONG-JONES: I think there's a lot more avenues and opportunities than there were 20 years ago.

Certainly, 20 years ago, you kind of really had to work hard to find yourself in an environment where you could have access to systems of a sufficient scale to be able to really see some of these failures happening in production.

So I think there are now college programs that enable you to actually get your hands dirty working with real systems that can potentially fail.

So I know, in particular, this is a thing that Mikey Dickerson-- yeah, Mikey Dickerson has put together a college program that he is teaching at Pomona and has kind of offered to other people to teach at colleges, to have people design and run a service that serves an RPC endpoint, and then he increases the kind of criteria as the semester goes on to really have you build a reliable system from first principles.

And I think that Mikey Dickerson's course is awesome.

So look for where that is offered and try to take advantage of that, because that is the most hands-on practical way of getting started with this stuff.

But also, summer internships are really great, especially if they are at organizations that are growing.

If you can get that internship at OpenAI, do, right, because that's going to be one of the most interesting companies of the next five, 10 years.

And they are experiencing scaling challenges that you wouldn't necessarily see anywhere else.

STEVE MCGHEE: Great. Thanks very much.

So just to sign off, is there anything else that you want to close with or tell our audience?

And, of course, where can people hear more from you and learn more about what you guys do at Honeycomb?

LIZ FONG-JONES: First of all, Honeycomb's website is honeycomb.io.

And my website is lizthegrey.com.

And yeah, I think, mainly, observability is something that is a spectrum, and it's something that you can aspire to get better at.

It's not a one-time event.

It's not a, I have it or don't.

It's something where just investing a little bit can yield outsized returns.

And I think that, going back to the SRE theme, it helps you achieve that goal of having better SLOs, of meeting those SLOs. STEVE MCGHEE: Excellent.

JORDAN GREENBERG: Awesome takeaway.

STEVE MCGHEE: Thank you, Liz.

Thanks for hanging out with us today.

LIZ FONG-JONES: It was super fun.

STEVE MCGHEE: It's been great to see you, and talk with you, and learn more about observability, which is, turns out, still evolving.

It always will be, I think. That's great.

All right. Thanks very much.

JORDAN GREENBERG: Thank you.

STEVE MCGHEE: Bye.

[JAVI BELTRAN, "TELEBOT"]

JORDAN GREENBERG: You've been listening to Prodcast, 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.