# Google Prodcast Season Three Episode Three

[TELEBOT"] You missed a page from telebot

STEVE: Welcome to season 3 of The Prodcast, Google's podcast about site reliability, engineering, and production software.

I'm your host, Steve McGee.

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.

(SINGING) --from telebot You missed a page from telebot You missed-- STEVE: Hey, everyone, Welcome to The Prodcast, Google's podcast about site reliability, engineering, and production software.

I'm your host, Steve McGee, as always.

This week, we have a special guest and a special co-host.

Jenn, can you introduce yourself?

JENN: Hello, I'm Jenn Petoff.

And I'm currently director of Google Cloud platform and technical infrastructure education.

So my area of expertise is in SRE education and driving a culture of reliability across Google and beyond.

And yeah, I've been at Google 17 years.

I'm one of the co-editors of the SRE book that we wrote back in 2016 and absolutely delighted to be here today as the special co-host as we talk to Google SRE OG Ben Treynor.

STEVE: Ben, can you introduce yourself?

You don't need an introduction, but maybe it'll help.

BEN: Apparently, I am the SRE OG.

Formally, I'm Ben Treynor Sloss.

I am the VP of 24/7 here at Google, which roughly means networking, data centers, and site reliability engineering for all of Google.

And I've been here, now, 21 years, which has seen a certain amount of growth and change.

And it has been fun to watch the SRE team and profession evolve to meet the changing needs and growing scale that Google has brought.

STEVE: Cool.

Well, Ben, I do remember, the first time I interacted with you was actually a long time ago in a Borgmon meeting.

And you were this guy who walked in and told people what to do.

And I was like, who's this guy?

And it turned out you were the boss, and I had no idea.

It was pretty funny to me at the time.

And that just kind of, maybe, dates me a little bit.

But so, this season on The Prodcast, we're focusing on software that SREs have built, or software, specifically, that we're focusing on less on the ops, and the toil, and the sysadmin stuff, and more on the building of systems, or improving of systems directly.

And Borgmon is one of those ones that I think of when I first think of big pieces of software that SREs contributed a lot to back in the day.

But to you, what were some of the most impactful pieces of software that you've seen come out of SRE at Google?

BEN: Well, you know, a lot of the tools that SRE has built are, by design, created by the team to replace activities that they had to do by hand with automation that could be improved and made more vigilant and more reliable than you can expect a single human to do.

But most of our problems with production services come from changing something, right?

My standard quip is if I really wanted Google to be super reliable, I would just shut down the source control management system and go take a vacation for a couple months because almost nothing would break.

**Things break because we change them.**

And so therefore, some of the most useful software that we've built are pieces of software that allow change to be propagated into a large, distributed, running system safely, right, and quickly.

So there are systems internally, like Sisyphus or annealing, that we use to make change to running systems without a high false negative rate-- i.e.

the system pushes a change that breaks things and doesn't notice-- but also without a high false positive rate, which is the system, you know, stops or rolls back a change because it thinks it may be breaking something when, in fact, it was totally fine.

And both of those then allow Google to run a higher feature velocity with the same level of safety and with a very low level of toil.

And so though there are many, many systems-- and this, in no way, diminishes many of the other things that we've built-- the ones that are focused around safe change management, for me, are very top of mind, mostly because when they don't work, then I get alerts, and phone, calls, and so on. And so that's a very--

STEVE: You hear about it.

BEN: --a very selfish perspective.

STEVE: Cool.

JEN: Sounds good.

So I think, in SRE, we tend to be very reflective.

We like to think about what we can learn from what we've done over the past number of years, et cetera.

I also like to think about what-ifs.

So this is a bit of a counterfactual warning.

You know, if we enter the bizarro world, what would Google have been like if SRE didn't happen as it did?

Like, was it inevitable for SRE to evolve the way it did or to be implemented the way it was?

And yeah, what would it be like if it hadn't?

BEN: Well, we actually have a lot of data points about how that plays out.

Because we have a number of other firms in industry who are successful, but have taken a different approach, right?

Famously, another large distributed services company has said SRE is the wrong answer.

That is not a good way to go because it separates the development team from the reality of the service they're running, and dot, dot, dot, dot, da-da-- totally reasonable perspective.

They've pursued, essentially, the DevOps approach.

And actually, we have this experience fairly regularly, where folks from that company then come to Google.

And when they join Google, they start out with SRE is the devil, right?

This is the worst thing that you can possibly do.

It's a terrible way to run services.

And it decouples the engineers and the services.

And we shouldn't do this.

We shouldn't have SRE teams.

And somewhere in the first 18 months, they all change their mind.

And I think it is mostly because they actually see that, yes, done poorly, separating the development team from their service actually does yield a very bad outcome.

But done skillfully, with the right incentives and with, very much, the SRE team being cooperative with, and in many ways, sort of behaving like a part of the development team, actually, it produces a better outcome.

You end up spending less time to produce a similar level of availability, in part because you learn the lessons from each service and apply them to every other service.

If you look at any of these organizations, they tend to be organized around the service or product that they're building, and they tend to be somewhat siloed.

And really, we see the same kinds of problems and the same kinds of needs cropping up over and over across many different services.

And so the ability to say, ah, we had a progressive rollout failure in this area, and we learned three new things about what we need to do around failure monitoring, we can now put that into a software platform, and all of Google services can benefit from, that's actually quite powerful.

And you know, the data, even from folks that come from other companies, is, hmm, now that I understand how this works, I'm spending a half to a third as much total development time to get a similar outcome here at Google.

Now I get it.

And they become our strongest proponents.

JENN: Yeah, so it sounds like people are won over once they kind of see it in practice when they join Google.

That said, I think no model is perfect.

So with the benefit of hindsight, which is 20/20, of course, what, if anything, would you have done differently?

Are there any tweaks you would have made now that-- knowing what you know after 20 years of SRE at Google?

BEN: Oh sure, I mean, every single outage brings with it a degree of regret.

I think the biggest single change is that we were slower than optimal to notice that we were building the same software systems in many different groups simultaneously.

And although it took, and still takes, a great deal of coordination and exhortation to get these different groups to cooperate on their software platform, it has yielded enormous results.

Like, the reliability has gotten better.

The toil levels have gone down.

And it paves the way for even more services to need much less operational engagement.

Because now, every new thing that we build can benefit from the platforms that we've already got to do monitoring, or change management, or capacity planning, or et cetera, et cetera.

And so I think, in retrospect, sort of realizing that you were going to end up with several different miniature production software stacks, each of which was independently owned and operated, if you will, but was not the same, and moving everyone to a common platform where they all could benefit from everything learned by the other groups, that we could have done that maybe five years earlier than we did.

STEVE: Cool.

Shifting into the present day a little bit, AI is cool.

I've heard it's kind of a thing.

There's two terms that I think get a little confused out in the world.

And maybe I can give my example, or my definition, and see if you agree.

But the question will soon follow.

One is MLOps, which, I say, is like building the ML itself, running the machinery, and like managing the data sets and the pipelines.

So out pops a ML model that you can then ask questions, or whatever you do with it.

And there's AIOps, which is, given one of these AI systems that's already been trained, how can you use that AI thing to do any other sort of operations?

You could even run it on like a traditional PHP LAMP stack or something.

I don't know-- but using AI to do ops.

So I think those definitions ring true.

But my question is, how has any of these-- how has ML, AI, more recently, Gen AI affected Google SRE?

BEN: Well, one thing to realize is that the two types of operations that you described, one of

them being-- let's call it training, broadly speaking, a creation of a model of weights.

And the other is serving, the use of a model with populated weights-- have pretty direct analogies to things that Google has already done.

Building a model is a lot like indexing the web.

You pull in a bunch of information.

You do a bunch of processing on it that has to occur once, in a central place.

And you generate this base model, right?

And serving, for machine learning, is, in many ways, similar to serving a web search index, where you take it, and you need to tweak it.

You need to pull in current information.

You need to worry about language settings.

You need the user's context and history to help get the most useful answers for them.

Serving somebody information about a dinner restaurant that is 6,000 miles away from them is much less useful than giving them one that's five miles away, et cetera, et cetera.

These have very direct analogies for what we see in machine learning.

And a lot of the ideas and principles that crop up, whether it's how do you deal with a batch pipeline when it comes to training, and what about checkpointing and restarting, and how do you deal with partial failures, and so on translate directly over into the challenges that we now see building and serving from machine learning population.

JEN: Of course, following up on that, there's like, how can we use these AI systems to improve our own craft-- so the practice of SRE, and using AI tools, and things of that sort.

Have you seen opportunities in that space so far?

Like, what are your-- are you feeling bullish about that?

BEN: Oh, extremely.

So in many ways, what machine learning is particularly good at is spotting subtle correlations, right, and making predictions based on subtle correlations.

OK, what is predicting failure, if not exactly that, right?

I've noticed that the following six monitoring signals, when combined, indicate we're having a problem.

You use relatively coarse measures to detect a problem with traditional monitoring.

A machine learning system can be much more sophisticated and subtle about it.

And it has the advantage of it never sleeps.

It never gets distracted.

It can be vigilant in a way that you can't expect a person to be vigilant.

And so there's this enormous opportunity, which we're already taking advantage of in a number of places, to use machine learning models to improve both our detection of problems and our differential diagnosis of problems, and in some places, not where you would expect.

Like, we detect electrical system impending failures in data centers now using these systems.

And it's very cool.

Because they have a characteristic pattern that they go through as they start to heat up, and their impedance changes, and so on.

And you can tell the difference between that and normal diurnal cycle operations.

But all of the applications around signal processing, for example, looking at latency, and error rate, and distribution of queries, and so on are all ripe territories.

Another totally different one, but super relevant, right, is, during an incident, one of the big challenges is somebody new joins the incident management, and they need to get up to speed on what's going on.

The incident has been going on for 20 minutes.

And you have this very long chat history, and separately, a whole bunch of monitoring signals.

Well, the generic idea of summarize for me, please, and you get back, in five seconds or two seconds, a paragraph that describes the most salient points that you need to know is incredibly helpful for someone.

And that time that they gain by not needing six minutes to read all of the background is six minutes less before they can actually be productive and effective in mitigating the problem.

So all of these are in use right now.

STEVE: That's cool.

And I bet the incident commander would prefer not to spend time writing up that summarization every time someone asks for one, or they need to send out a post to external comms, or something like that, too, for sure.

BEN: **I look forward to the first time that some unsuspecting executive pops onto a chat to ask what's going on with their service, and they get back a Gemini-generated response.**

STEVE: Nice.

BEN: I think that may be an aha moment for everybody.

STEVE: Cool.

Can we call it OMG bot or something cool like that?

I think that would be fun.

BEN: There's probably other terms that we could use.

The executive dispensary bot might be a good one.

JENN: Executroid, yeah.

BEN: Though the snark-- setting aside the snark, the ability to bring people up to speed with information that's relevant to them-- setting aside the exec use case, right, if you're a support engineer, you need a different set of information than if you're an SRE working on a subcomponent of the service.

All that information is in the combined data we've got, but you need to summarize it in a very different way.

Right now, you had two choices.

One, ask somebody who already knows all this stuff to write you the summary.

Number two, read through everything, and figure it out yourself.

This opportunity, which is-- I mean, it's a lob serve, right-- actually solves this problem really nicely.

And again, that speed that it can do it, which is much faster than a human, if you need it, as a support engineer, to write a customer-facing summary of what's going on, having an AI system generate the first draft in seconds is five minutes less before all of your customers can be informed about what's occurring.

STEVE: Yeah, similarly for fixes as well, write me some YAML to do the thing.

And I wouldn't submit the YAML directly myself, personally.

But at least, it gives me a head start.

BEN: Yeah, right, you might be three times as fast.

STEVE: Right. Cool.

So in my job, I speak with cloud customers all the time about adopting SRE practices, and what's the difference between DevOps and SRE, and all these words, and phrases, and things.

But one of the things that comes up a lot is, we want to build an SRE function.

Do we need a Ben, basically?

Like, do we need a head of SRE?

We've kind of jokingly called this a CRO, Chief Reliability Officer.

I think that's a little much, but you know, I don't know.

But really, do you think it helps to have a seat at the table representing reliability, regardless of title?

BEN: I think it does, in much the same way that we see a lot of organizations gravitate toward having a chief information security officer.

And there's a lot of reasons, but one of them is the domain is pretty deep.

So having real expertise in it turns out to be important.

And another is that it's important, but only occasionally urgent.

And it is very easy, in an organization, to get distracted by all of the urgent things.

We have a new customer need.

There's this feature.

We have this deal, dot, dot, dot, dot, dot, right?

And gradually, because you don't have outages very often, you spend less and less time on the very important topic of making your service reliable until you start having a string of outages.

And then, it's 18 months to rebuild, if you're lucky.

The segmentation of resources, time, attention, and expertise has, in practice, worked very well for us.

And I've seen it work very well for a lot of other companies.

It's not the only way to do things.

But if you want to build an SRE team that takes advantage of, for example, the cross-service learnings, the ability to start in incorporating risk measures, I think it is very helpful to have a lead for it.

STEVE: Cool.

JEN: So I think now, we can maybe change gears again a little bit.

And we talked, in the intro, a little bit about the SRE book, which, as an aside, for me-- coolest project, hands down, without a doubt, that I've worked on in my 17 years at Google.

So Ben, thank you for giving us the opportunity to write the book and giving me the opportunity to basically herd the cats, so to speak, to ensure that the book was written, bringing together the voices of 70 contributors representing over 500 years of Google production experience distilled down into the tome.

So the SRE book arguably accelerated SRE adoption at other companies that are outside of Google.

And again, you talk to a lot of Google customers, et cetera.

Which ideas from other implementations of SRE have you seen out in the world?

Have you seen ideas from those implementations that Google has adopted, or perhaps should be adopting?

Like, what can we learn from others?

BEN: Yeah, we had, maybe three years after the book's publication, gotten together a few groups of folks who were their SRE lead in their company and talked to them about how they had used what we published and what they had learned about running SRE in their organization, to good effect.

And in more than a few cases, they had come up with interesting, novel adaptations.

A lot of them fell into the category of-- you know, the specific way that we've done SRE at Google is a reflection of both the underlying engineering principles, but also how Google is organized and managed.

And other companies have very different organizations, very different management, and very different priorities in some cases.

And so you see the ability to adopt this idea, like, I want the development team to have a strong, personal incentive in making a service that doesn't require much human tending.

OK, at Google, you do that by saying, we share headcount, right?

If we need one more headcount in SRE, then you have one less headcount in development.

Great, now we have a shared goal to need as few SREs as possible.

And that guides us to an efficient allocation of engineering time being spent on making the service reliable by default.

Other companies may budget things totally differently.

And in there, if you get the principle, which is, but I still want the development team to have real skin in the game, as it were, then one approach would be to say, OK, well, at our company, right, what we do is we don't have enough SREs for everybody.

And so we pick and choose which services SREs work on and which services get the DevOps treatment based on the effective yield, right?

Essentially, which of these services needs us the least?

Because those are going to be the ones we're most efficient at working with.

The fewest SREs can make the most difference for the company.

And that is just a different mechanism to guide development teams to build a service that doesn't need much human tending and doesn't need much toil.

But the idea is, there, I'm giving you a management perspective.

There are also definitely technical perspectives.

But you know, you ask a VP, you get VP answers, right?

But these management techniques of getting these different groups to do the thing that's actually important and useful for the company and apply the principles from SRE to the instantiation of what that company actually needs, those have been both useful and gratifying to see.

STEVE: Great.

JENN: Very cool.

Well, following on from that, again, we talked about, at the beginning-- so my area of expertise is sort of education and SRE education.

Should we teach SRE in college, university?

BEN: Should?

I will admit, I was surprised by how much interest there was in SRE.

When you and the others approached me to write the book, I thought it would be fine, and it would be a way for us to talk a bit.

I thought we might-- might-- see 3,000 copies out there, right?

So I was off by two orders of magnitude in my estimation of how much interest there is.

I'm similarly thinking, oh, college, OK.

But you're just learning computer science principles in college.

Is the idea of giant, distributed systems, and progressive rollout, and error budgets-- is that going to resonate?

Or is it too early in your career to really grok why those things are interesting?

I don't know.

I've been overly pessimistic about the level of interest in utility for SRE before.

I certainly wouldn't want to make that same mistake again.

JEN: Maybe there's some happy medium.

Because I think, at universities today, people think about things on a very small scale, a very local scale.

So even just getting a little bit of a dose of how do you scale things up-- like, what's going to be different if you choose a career in industry?

And the other thing that's interesting, there have been a couple forays in this space.

Like, Mikey Dickerson recently tried a course at Pomona.

He talked about it at SREcon, and it was fascinating.

He basically had his students build and tend a service through the course of a semester.

And basically, they were graded at the end of the course based on the uptime that they were able to achieve.

Like, did they meet their SLO?

Did they not meet their SLO?

And I thought that was really a fascinating approach.

And we've tried a couple of different approaches, even with Google SREs.

Like, one of our colleagues, Christoph Leng, has taught a master's level course in Germany.

And we've kind of put a toe in the water with just some online courseware.

So interest is definitely there.

But yeah, is it a nice to have elective, or is it something that's really foundational?

I guess it remains to be seen.

BEN: And it might be that the right medium is to have some of the lectures on the basics of managing a big, distributed system and how the production management practices are done, you know, best practices.

Because certainly, learning things like progressive rollout and rapid differential diagnosis, those are concepts that every software engineer should have.

I mean, as we've said, our ideal is every SWE spends six months in an SRE team somewhere before they go off and work on other services because that's a bit of background for them.

And so there may be a real opportunity to teach these concepts and a little bit of practical application for everybody going through, let's say, a computer science curriculum.

STEVE: Cool.

JENN: Very cool.

Well, Steve, do you have-- so I think we're nearing the big finish.

We have a big question to ask you at the end, Ben.

But Steve, do you have any other pressing topics you want to tackle while we've got Ben's attention?

STEVE: No, let's do it.

Let's dive into the final one.

And Ben, you'll never see this one coming, but what's it going to be like in the future?

Like, please predict the future for us, which, we all know, is perfect.

What's SRE going to be like in the next two years, or heck, the next 20 years?

Like, how is this all going to play out, according to your prescience?

BEN: So I can assure you, it won't be perfect.

STEVE: Right. Darn.

BEN: The trick is any competitive industry is going to always be pushing the boundaries.

And so you're always going to be trading off things like reliability, risk, efficiency, feature velocity, right, scale.

The goal for SRE is that you use technology and engineering to improve the efficient horizon of what's possible so that the same size group of people can get more feature velocity, more scale, fewer outages with the same number of hours invested for their services over time.

Because they're taking things that had to be done by hand, or that introduced risk, and they're replacing them with technology that takes away that need.

So I do think one of the most interesting, and for me, exciting frontiers that we're pushing right now is the move into risk assessment.

There's this STPA framework, now, that we've been working with for several years.

And the team, along with the group at MIT, has made quite a bit of progress in making STPA far easier to assess.

And for those not familiar, STPA, you run a process, mostly with people.

And it spits out a set of vulnerabilities and risks that your system has.

And then, you can choose to fix those things.

And it's very, very good and thorough at it, but it's also time-intensive.

STEVE: Yeah, it's Nancy Leveson work at MIT, I think, right?

BEN: Yeah.

But we're getting, with them, much, much better at making it easy to do-- a lot less time invested, some tools that start helping you do it.

I mean, it's, in some sense, following the normal SRE path of a thing starts out being manual, and then we figure out how to make it easier, and we figure out how to introduce technology to do pieces of it, and da, da, da.

And the idea of I can tell you all of the outages that you're going to have before they happen, and then you can decide how many of them you want to fix and how many of them you want to live with as sort of an engineering choice, that's incredibly interesting.

Because right now, we use heuristics to do this.

But we can use actual assessment to do it.

That's two years' time frame.

I think 20 years, I am going to not make the mistake that several other notable industry figures have made of trying to offer a prediction, on record, for what 20 years from now looks like.

It is-- the current machine learning revolution indicates just how wrong you can be with that.

I do-- I will put a stake in the ground and say I think the innovation pace that we're seeing is likely to continue.

We don't-- despite the end of Moore's law, for example, we don't really see that the pace of innovation in services, or technology, or even ideas is slowing down.

And if you're wondering what I mean, go look at what the world looked like in 2004.

Smartphones?

I mean, the world, now, in technology, is unrecognizably different than it was 20 years ago.

I expect it will be unrecognizably different 20 years from now as well.

And that's part of what, for me, makes this such a fun industry to work in.

STEVE: Great.

Well, thanks a lot, Ben.

This has been a great conversation, of course.

Thank you, also, to Jenn for helping me out today with co-hosting duties.

JENN: It's an honor to be here.

STEVE: Before we go, is there anything else that you want to throw in there, Ben, like a pitch, or a landing page of some kind, or just your favorite GIF, maybe?

BEN: My favorite GIF?

JEN: GIF or "JIF?"

BEN: Yes, yes, the eternal debate.

I think-- well, what I said briefly, in passing, earlier on I actually stand by.

**I think if you're working in software or you're interested in working in software, it is super useful to make a point of spending a few months working directly on production problems.**

And whether that's-- if you're lucky enough to have an SRE function at your company, working there.

But if you're not, getting your hands dirty, and working on the system, working on production problems, seeing how things work and how they don't work, I've only ever seen that background be incredibly useful for people who subsequently go on to try to build the things.

And I liken it a bit to, you know, if you're going to work on cars, you should know how to drive.

And a similar, if you're going to work on software systems, you should understand how they function at the ground level.

Because it lets you build systems that will work in practice and let you avoid some of the hazards and pitfalls that come with building large, successful services.

And if you build a service, you kind of want it to be successful.

If it's successful, you'll have the problems of scale.

SRE is very much about solving the problems of scale without the investment of scale, if you will.

So it's really useful background.

STEVE: Great.

Thanks to you both.

Hope you have a nice rest of your day.

And maybe we'll see you again in season 4.

BEN: Great.

Thank you very much.

[JAVI BELTRAN, "TELEBOT"] You missed a page from telebot You missed-- PERSON: You've been listening to Prodcast, Google's podcast on site reliability engineering.

(SINGING) Telebot, telebot

PERSON: Visit us on the web at sre.google, where you can find papers, workshops, videos, and more about SRE.

(SINGING) You missed a page from-- PERSON: This season's host is Steve McGhee, with contributions from Jordan Greenberg and Florian Rathgeber.

(SINGING) You missed a-- PERSON: The Prodcast is produced by Paul Guglielmino, Sunny Hsiao, and Salim Virj.

The podcast theme is Telebot by Javi Beltran.

(SINGING) Telebot PERSON: Special thanks to MP English and Jenn Petoff.

(SINGING) Telebot, telebot, telebot