Prodcast Season 5 Episode 8

Guest: Damion Yates, Google DeepMind
Hosts: Matthew Siegler, Steve McGhee

[THEME MUSIC]

SPEAKER: Welcome to season five of *The Prodcast,* Google's podcast about site reliability engineering and production software. This season, we are continuing our theme of friends and trends. It's all about what's coming up in the SRE space, from new technology to modernizing processes. And of course, the most important part is the friends we made along the way. So happy listening and may all your incidents be novel.

STEVE MCGHEE: Hey, everyone. Welcome back to *The Prodcast,* Google's podcast on SRE and production engineering. I am in the morning in the US. And our guests are in the evening in a different part of the world, I think. Florian, you're here. How's it going, man?
FLORIAN RATHGEBER: Hey, everyone. Good to be back. My name is Florian. And I'm calling in from Switzerland.
STEVE MCGHEE: Oh, man. It must be a whole different time zone over there. Is that-- is that true? Or is that just a myth?
FLORIAN RATHGEBER: That could be true unless there is only one time zone, and we've all been lied to.
STEVE MCGHEE: I think the Earth might be flat, but I'm not sure. And we have another guest in a whole different time zone kind of between us, yeah, pretty much between us. Dear secret guest, who are you exactly? And where in the world are you?
DAMION YATES: So I'm Damion, Damion Yates. And I'm based in London, have been the whole time I've been working at Google. And, yes, it's pitch black outside, different time zone.
STEVE MCGHEE: OK, cool. So you're on the round Earth part of the theory? Is that right? You don't subscribe to the flat Earth, no time zones theory that we just came up with?
DAMION YATES: I do think that we should have just BST and everyone just sort of has a concept of going to work at 2:00 in the morning in certain countries, or 8:00 PM. It would make commuting much easier if we just had one time zone, but you just change your times.
STEVE MCGHEE: Let's just do metric UTC. Let's just do it all.
FLORIAN RATHGEBER: Yeah, no.
DAMION YATES: Yeah metric and UTC. Well, no GMT, but let's point out that Greenwich is where time started.
STEVE MCGHEE: Classic, cool, well, thanks for coming on *The Prodcast.* This is going to be a fun one. Actually, Damion and I worked together a long time ago. And by a long time ago, I mean, at least some unnamed number of years ago in the London office. So it's nice to see you again. What are you doing now, man? What group are you--
DAMION YATES: I think it's 19 years.

STEVE MCGHEE: Listen, we don't need to count numbers. This is unnecessary. But where are you now? Tell us which part of the company you're in now. What is it that you're doing?
DAMION YATES: So I'm in what is now known as Google DeepMind. I moved into DeepMind, it'll be 10 years in a month or so from a department aligned with where you were. And in the last 2 and 1/2, 3 years after merging with a different part of a Google Brain, Google DeepMind was formed. And I've been there since.
STEVE MCGHEE: So when you joined, it was just called DeepMind. And I think was it a separate company but under Alphabet or something like this? Is that how it went? Or does it matter? Maybe it doesn't matter.
DAMION YATES: That's complicated and spicy.
STEVE MCGHEE: Fair.
DAMION YATES: Let's just say not really, but we looked at becoming quite a separate bet, and then we didn't. And now we're definitely just a bit of Google, but we're a different unit. And I don't think it's interesting, but we're not classified separately.
STEVE MCGHEE: Gotcha. Certainly not now, now we're all happy, friendly family.
STEVE MCGHEE: One really important question, have you played chess with Demis yet?
DAMION YATES: I have not played chess with Demis. I know the rules. I know how the horse and castle move. But one of my team did win the chess championship recently at an off-site. Because Demis was off doing multiplayer, when you play lots and lots of people at the same time? So there were people doing that. And he took the opportunity to do a competition and won. I'm really proud of him.
STEVE MCGHEE: That's excellent. Very cool. From what I remember, when you started, there was not a lot of SRE going on in DeepMind. In fact, I think you would say it was almost more of a laboratory than a production-- there was no Gmail-sized service that you needed to support or anything like that. Is that right?
DAMION YATES: Yes, it is quite an interesting beginning. So DeepMind were acquired, I think it's 12 years ago. They formed two years prior to that. And they did have some ex-Googlers, some of whom were in the SRE org inside Google, join the team. But they were typically working on various bits of infrastructure related to whatever training or machine learning research that was taking place.
So there was no coordinated reliability engineering model for the company. And I was brought in initially into a team called the systems team, who ran a data center external to the Google ones for some of the legacy DeepMind research that had taken place before acquisition.
And that team had been there from the start and looked after their own accelerators in a data center, lots of machines. But they weren't familiar with the Google way of doing things and hired me into that team. And so there was a lot of use of the Google infra, but there was nobody coordinating and looking at any reliability engineering side of things, or best practice, or any concept of resilience.
It was, for example, whenever a new resource came along, or if you had an entire other data center available to you in the Google infra, rather than build some sort of resilience between your experiments across one and the other, they just doubled all their capacity and did more experimentation. So it was just more and more points of failure.
So the beginning was entertaining. And I essentially set up the idea of a reliability engineering team in an org that didn't have any public services, didn't serve to the public, and an outage there would mean, we would-- Demis told us AGI was 10 years out. And if we were down for an hour, then it was 10 years plus an hour.
So it didn't seem that critical. But there were keen researchers, their entire day job was to run experiments and do this. And nobody could work if there were situations that caused that. So it

was quite immediately obvious that this was a valid thing to exist. And I think it took both me and DeepMind a bit of a surprise, like, I can't believe we didn't do this before. And so things grew from there. I got a whole team and churned forward, building a sort of an infra behind providing increased reliability to experimentation.
STEVE MCGHEE: That's cool.
FLORIAN RATHGEBER: So effectively, your initial job was introducing reliability concepts at a research org, as you said. So compared to how you did things before, what were the changes? Or how did you need to change your approach, I suppose?
DAMION YATES: So what I will first say is I wasn't brought in exactly to do that. I determined that was the best way forward. It was more like an xy problem. Damion, can you come in, and just increase quota when we keep running out? And I, initially of course, what's the expression? When you're constantly playing catch up, or you're reactively fixing things as they occur, that is what I would do.
They didn't know the commands to increase the resource availability. And so I realized very quickly that instead of being the case that a researcher tries to launch an experiment on the Google infrastructure and is told-- it returns back and says no more GPUs or TPUs, rather than them just waving their hand to the 100 other people in the company at the time and saying this place is full, let's start using the other one, which was just embarrassingly poor way of running things, I measured utilization-- added tooling to measure utilization of this, had warning emails when things were approaching capacity, and realized that this could be turned into an entire discipline.
So from the start, for the first six months or so, I was just useful. And everyone was like, oh, it's really useful. We got someone to unblock us. But I also was like, this is just unsustainable and silly. And we could do a much better job of this and not run out.
And so I built a monitoring infrastructure. Maybe it's a little bit too much detail, but the accelerators in the infrastructure, they were newer. And they weren't measured as much. The sort of existing tooling wasn't there. So I had to build from the ground up. The rest of Google's infrastructure was absolutely rock hard and had been there for a decade or more when I joined. But the accelerators were a sort of a new element to this. And so it was sort of a little bit of a forgotten resource. The dashboards to go and see how your job was running would show memory utilization and CPU utilization, and then there'd be nothing for how much GPU you were using. So I found ways to pull those stats and then built a system around that.
Florian, you mentioned enthusiastically setting up a sort of reliability mindset. That was an economy of scale thing I came up with. I realized that-- it's like the whole teach people to fish rather than do the fishing for them idea.
I built some training courses. I made them optional and didn't get very many applicants. I wanted to explain, to have this mindset of-- for example, you're running an experiment, you want to query out to a system and grab back some results and then carry on processing. And the code was written just to request. And if it didn't work, the thing just crashed and gave up.
And I said, these services are-- they're designed for en masse, large Google bits of infra-- Search, Gmail, et cetera. And they deal with failure. And they'll retry. They can't always guarantee full availability. And all you need to do is a retry on your experiment. And it doesn't crash. And you have to wait for it to restart, et cetera.
And they're like, oh, that's a genius idea. And I'm like, it seems straightforward to a reliability engineer, but this is the influence I was able to have as the company grew and grew and my team was small, it became training courses, little bits of influence, engagements. So at scale, yes, slowly, trying to encourage this mindset worked well.
STEVE MCGHEE: That was going to be my question. Did you do trainings? Or was it just like

you being helpful and people just figured it out? Or were you explicit? Also, for our guests, our guests are SRE who don't work at Google. So they are folks out in the world. And so they might be experiencing something like this.
How did you-- I don't want to say like exert your influence, exactly-- but how did you get this secret skill into the brains of everyone around you, who it wasn't their job? But you needed to somehow stick it in their heads? What worked for you, especially in this research-oriented organization?
DAMION YATES: So there were multiple techniques. I think one of the best ones I came up with was to talk to the new starter team and add the gist of the infrastructure, how to run production things on this system. So when any engineer or any tech role, so a research scientist or a software engineer, research engineer started, and maybe a technical program manager, on those particular roles, the new starter team would give them a big curriculum of training courses. And mine was slotted in there quite early on.
We eventually, after a few years, had a tool for launching experiments which simplified and took the friction and edge off dealing directly with how to launch on this. And it would help, before that call, I said do mine first, so that they know what they're running on.
So for example, understanding the concept of containerization and things like that in terms of your limits and being mindful of network capacity limitations if you-- or the size of machines. So a research scientist may want to just increase the number of CPUs, make the thing go quicker and quicker. And I would explain that you really want to be scaling horizontally, have multiple instances and connect between them. But be mindful of the fact that you're going to be limited. There are priorities for network traffic and things, distance, locality. These are all settings. I wouldn't give them all the information in the one hour training. I would say, bear in mind that this is a concept you can go and look up or ping me later. And I can point you at docs to increase your quality of service, flag settings to increase the likelihood of your job communicating quicker or constrained, if you like, closer to the same rack, so that the network works better for them.
I will just throw in, actually, the defaults in the Google infrastructure are to spread it a little bit for domain failure and such, like in any given cell, so you can ignore that completely if you're happy that your experiment will just fail to work if there's a power outage in your rack. But that's fine. I've mentioned, you're delaying the onset of AGI by 10 years and plus an hour or two that your things out. But you do want it to function as quickly as possible when it's running. So we would do things that are not normally best practice in the typical serving style, but I would optimize for speed for their experimentation. So that was one was getting the new starters to have their-- slotting my training in there.
The other one was-- I didn't complete this program, but we were planning to roll through the whole company
DAMION YATES: We had the concept of a training on scale. If you wanted your experiment to grow, you could just add some zeros to the replica count and make the thing absolutely huge. But I wanted people to realize that although we had touted this as huge amount of infrastructure, we are using hilariously large amounts of resource. And somebody upping their replica count can have knock-on effects.
So if they notice that everything's running slowly, and then they're getting pings from SREs in different teams going, is that-- did you start this? I've said stop your job. Yours is just an-- if you just started it, stop it again. And if everyone goes oh, that's better, then leave it stopped and talk to my team. Talk to an SRE in DeepMind, reliability engineer.
If stopping it had no impact, then the SRE in, say, Gmail was wrong, and they maybe just picked on somebody. And their experiment can carry on. These experiments were important to them. It was their day job.

So I wanted to run through and give a sort of running on infrastructure license. And then we would gate access to the group that would allow you to launch jobs. So I was going to roll that through. We slowly started to collect names. But the company grew faster than I could do training. So we'd be getting new starters who'd gone through this.
And I think as we grew, we'd gained, I'd say enough software engineers rather than research scientists that there was already a semblance of awareness of building for scale, building for resilience. There was a growing improvement, I would say, that decreased the necessity for me to enforce a training.
STEVE MCGHEE: Cool, that's good.
DAMION YATES: I think it was fairly common for you to maybe have-- maybe you'd have a local GPU in your machine. Or you might, in the lab, have potentially dozens, maybe 100 machines. Let's say you could have some way of connecting to-- I'm trying to remember the various-- Slurm, something like that, some sort of external product to allow scheduling.
And they were used to-- you'd launch and run. And an outage might occur because of power network, or crash, or something. But you were pretty much the only user at your time in the lab. Or you may have agreed verbally with someone next to you who's using 50 of the machines, you're using the other 50.
And at the scale that Google run at, it's like confidentially large numbers of machines. And it has to be more fungible. You have to share and move. And so weirdly, it's less reliable. And you should expect failure.
So one thing we taught heavily in our course was to checkpoint your progress, assume failure. And this will help you out. It is difficult to build in checkpointing code, pick all the variables your thing is persisting, and reliably have it come back.
This is quite rare, but I do remember some people admitting I'm just not going to do it. I will just hope it works. If it doesn't work, I'll run it again.
STEVE MCGHEE: Damion, you told us something very funny recently, where you talked about luck. Can you tell us your line about luck? I loved it.
DAMION YATES: Well, OK, that's-- so earlier on, I mentioned that I didn't want us to work reactively. And if there's an outage, we fix it, and everyone's clapping. That, to me, that's a failure. As a reliability engineer, I'm not doing my job well enough if I haven't had telemetry tell me in advance and then I fix it silently, they don't know.
The downside to that, and I'm sure this will ring true for many people, is that you doing your job really well, makes you completely invisible. There's no impact at all. There's an avoidance of huge negative impact, but there's no outages is you going yay. And everyone else going, what team?
And so I've used the expression for years that I've said luck is our enemy. I don't like luck, because if a team builds something, it's not-- by the way, it's not all experimental. I mentioned the experiment launching system that was built. So there are services that run.
And if they only build features. And they don't build any mechanisms behind it for monitoring, logging, debugging, or resilience, and auto fixing, and it just functions, if they're really lucky, it will just be fine. And when I'm claiming, saying, maybe you should consider x and y to fix this, they're like, why? It's perfectly OK? So yeah, I don't know if this is funny. It's just sad that luck can be a really bad thing.
STEVE MCGHEE: It can be both. Yeah, I think it's funny and sad.
FLORIAN RATHGEBER: It can be.
DAMION YATES: Luck is what you want when you know it's about to fall apart. I've worked somewhere previously that had a machine that had been up for, I think, 11 years in Telehouse Liverpool Street in London. I can't believe they managed to retain power for 11 years. So this

machine had never rebooted.
Now, that's obviously a security concern and other things. But the disk had died ages ago. It was throwing SCSI errors left and right. But it was running in RAM. And it was acting as the main pack server for proxy configuration for the entire company. And there wasn't a fix that wouldn't have required an outage. So yeah, I love luck in some situations. So crossing fingers is a part of the role, I guess.
FLORIAN RATHGEBER: Yeah, I also had at a previous job, I discovered that there was-- several months after a colleague had left, there was still a task running they had started on a GPU also because the machine had just been sitting there idle for that long time.
But you already teased what I was going to ask next, which is the different jobs that you and your team were sort of taking on as time evolves. So you started with a lot of training, but then you also mentioned that you were building this experiment launching tool, so there's feature development, there's training, there's advocacy. What other hats do reliability engineers at DeepMind wear?
DAMION YATES: Oh, I like this question. So from the beginning, the company was quite small. I can't remember the exact number, something like 100, 150, maybe 200. It depends. I'd have to go and look to check.
So the company was small enough that most people either did things themselves following some documentation, or it just wasn't happening in any sort of conformed way. For example, any consideration around group management, or security, or as I mentioned, the ceilings resource, the quotas needed for resource allocation was very ad hoc and broken.
And so taking that just made sense and was quite scalable for a small size. As we grew, the group management, which production group you're in, we'd set this up with an automated internal system that, I suppose, it checks the configurations in. There's tooling that allows you to configure group memberships in a semi-automated way.
And you run your own stack to keep that functioning. And it pulls sort of configs to know whether to adjust groups. So we took ownership of that. There's the consultation side that I'd mentioned, the training. There was any sort of security incident, we did handle some escalations. Sometimes at night, we join-- what do they call it? A panic room video conference and discuss replacing lots of libraries if there's some sort of famous outbreak.
Google do a lot of stuff in-house. Most of the stuff we write is our own. But there's occasional use of certain things. And we would be involved heavily in security incidents, so security, group management.
Let me think, maybe, I can't easily enumerate the entire list off the top of my head, but yeah, we-- oh, what would the non-Google term for this be? The allocation of resources to people. We didn't have a compute planning team initially, so we basically had that.
And yes, there was a very big list of this. And it carried on after the company grew and grew. And actually, to this day, we're in 2026, and we're still receiving bugs for things that there's another team for this now. For example, we have at least 50 person strong security team. And I think we still get things that are kind of that team's role.
And that team grew. And I'm like, oh, there's now a security team. And I've contacted them, and they're like, yeah, that does sound the thing we're doing, but we're a bit busy. We're doing this thing with-- we're using AI to build security, Damion. But your team seem to be coping. We're like five people. You're 50. Yeah, yeah, yeah, but it's still-- so we're struggling to move everything off.
But the hats, yeah, there's a lot of hats in the team. And there's an element of heroism, I guess. Let's go back to that sort of reactive thing. That's not an ideal way of running a reliability engineering team, by reactively fixing things. You need to be able to scale. And you need to--

sometimes, letting things fail is OK.
Again, like I've mentioned, with luck, an outage, visible outages, much as I'm upset about it, when we fix it, we look amazing. They give us a bonus. I'm like, no, you're doing it wrong. But OK, I'll take the money.
STEVE MCGHEE: Not a good plan.
DAMION YATES: And then, of course, they're like, oh, do you want more headcount so this doesn't happen again? I'm like, yes, finally. So outages can be a good thing. Decreasing heroism. There is a Google thing with no heroes. That's actually a good idea. And it remains, to this day, a difficult thing to get to that mentality. But yeah, maybe we should-- maybe we should let things not work and no longer be the-- have so many hats.
FLORIAN RATHGEBER: Yeah, and outages are always learning opportunities as well, so yeah.
STEVE MCGHEE: So given all that allowing things to fail a little bit, to some level, within the SRE community, we talk about SLOs and an error budget. And like we know we don't aim for 100%. And you can allow some things.
But one of the most interesting things that I've heard talking to you in the past and in this conversation is that you're in a research organization. And as you've mentioned already, some of the things you were working on were like batch jobs. And they're allowed to-- they don't need to be up, up, up all the time, 100%, which is good. We want to have that flexibility, like we said. But at what point, how did you make that distinction of this stuff over here, quote, "gets SRE support" and this stuff over here doesn't, or didn't, or shouldn't, or won't in the future? Do you, as DeepMind, do have a demarcation line that you can distinguish these things? Or is it like luck of the draw? Or is it whoever is best friends with Damion? Or how does it work? Are there things that don't need to be managed by the SRE team?
DAMION YATES: So due to scale, I mean, I think there's 6,000 people now, and we haven't grown very much. So we had to prioritize. And I did come up with a categorization for this. Obviously, Gemini is critically important. And our team support the training of that.
Now, there's an importance level there. It isn't serving to the public. I think Google instant, everything you tap onto a keyboard these days is looking up search results for you. And if that doesn't work, they have graceful degradation. It just waits for the whole string and sends it. But there's levels of criticality, but there's importance as well.
And the style most language models use for training is kind of everything in lockstep. So this will horrify SREs who aren't familiar with this. But instead of having more machines meaning more resilience and some sort of load balancing and failover, it's every single one is critical, and if one of them goes slow or stops, everything has to wait. So it's worse when you make it bigger.
And the size of these jobs is confidentially large. And so an outage there is, in fact, quite serious. I mentioned fungibility earlier, and you might think other jobs could start. But actually, you can have that, but there's delays.
It's quicker to get a machine to reboot and come back, or a job to just be restarted and get everything in lockstep moving again than it is for lots of things to move out of the way and someone else's experiment of lower importance to start. So we tend to have the threshold set to give us an hour to get back, and so other people can't use this space, which means if you have an hour outage of a large number of chips, is large enough that it's like one of the actual serving production systems that people look after.
So there's a cost factor in size of job. So the engagement, whether we engage criteria of importance, leadership saying it's important because we don't always know. I don't know what the latest, greatest thing is. So we need to be told that. That's the number one thing.
If the team involved has themselves tried quite hard to engineer reliability into their system, but they just need help and guidance, they've won brownie points, if you like. And we will favor them

over a team that has not bothered at all, just on features, and then wants to throw over a fence or hand-wave away, your team are SRE, can you come and just build monitoring for our system? They don't score very well on the ranking.
We do have to prioritize. And that's not a great move. We have still engaged in those situations. We've given them a headcount that didn't do the work, and then very much spent the entire time teaching them to fish.
And this engagement actually went very well. They were very pleased with the outcome, because members of their SWE team were like, OK, we can do this. It's not that difficult. And it should be treated as a feature. Your system being up should be one of the criteria of top features, rather than just assuming it's up, and relying on luck, and then having the features people asked for in its way.
So yeah, there's limited team size limits our engagement options through the company of this size now. We're in DeepMind itself, so the role we're doing is basically the same as an SRE style role. But we call ourselves the reliability engineering team.
And there is a-- Google have SRE teams that would look after elements that are made use of for training, the infrastructure itself has underlying engineers that look after that, that we would liaise with. That experiment launching system, that was built by my adjacent team that I reported within the-- we built a platform.
It was known for a while as the platform team. That was sort of when GA became generally available and then gained a Google SRE team to look after it. So there's actual Google org SREs involved. And they're recently now, even more involved looking after some of the serving side of things.
The actual serving of Gemini, my team don't do that at all. But we work with the Gemini infra team who've built this lockstep infra. And we support that side of things.
STEVE MCGHEE: That's cool.
DAMION YATES: So yeah, the prioritization of engagement is complicated, and I think has been for a while. We've dealt with this, as I mentioned, with a high value, low contact things like a training course and advice rather than choosing to on board. There's less of that. But we have looked after service systems-- serving systems a little bit better over the years as well.
STEVE MCGHEE: Cool.
FLORIAN RATHGEBER: Yeah, cool. Well, a storied history and lots of useful advice and interesting anecdotes you've shared with us. So I think that was really awesome. I think our listeners will appreciate the insight into reliability in a research org, because that's probably a little bit unusual, especially at the scale, of course, that DeepMind operates at.
DAMION YATES: Can I just throw in that-- I mentioned chips functioning being very expensive. A few years ago, I was told, Damion, our most expensive resource are research scientists. Now, I'm not saying they're paid a lot of money, but even them not being able to build code, so like a test failure would sometimes come under our domain. That's another hat we sort of shared to an aligning EngProd team.
So I want to mention that because it's a key difference between research-- a research organization and a live serving thing is look at your most expensive resource. And for many years, that was a team of people pushing towards building what you have today. If you've used Gemini, that is years of research. And those researchers needed to not be just twiddling thumbs because they couldn't build, or they couldn't launch for outages and capacity things. So yeah, importance comes into this. It's not just is it up instantly like Search? But yeah, that's a thing.
FLORIAN RATHGEBER: No, no, totally. I guess that's the nature of a system that only serves internal users, that obviously, these users are also paid by your organization. So you want to keep them as productive as you can.

STEVE MCGHEE: Yeah, totally.
DAMION YATES: Indeed.
STEVE MCGHEE: Well, thank you, Damion. This was enlightening, as I'd hoped. No surprise there. And I'm sorry that this California kid has kept both of you awake on a Friday night, but I think it was-- it was worthwhile. Thank you very much for hanging out with us for however long that was. It was a good conversation.
DAMION YATES: Absolutely fine.
STEVE MCGHEE: All right, cool.
FLORIAN RATHGEBER: Totally worth it. Damion, thanks for joining us. Thanks for sharing your wisdom with our listeners. And looking forward to future episodes of *The Prodcast.*
STEVE MCGHEE: All right, thanks a lot.
DAMION YATES: Thank you very much.
SPEAKER: You've been listening to *The Prodcast,* Google's podcast on site reliability engineering. Visit us on the web at SRE.Google, where you can find books, papers, workshops, videos, and more about SRE. This season is brought to you by our hosts, Jordan Greenberg, Steve McGhee, Florian Rathgeber, and Matt Siegler, with contributions from many SREs behind the scenes. *The Prodcast* is produced by Paul Guglielmino and Salim Virgi. The *Prodcast* theme is "Telebot" by Javi Beltran and Jordan Greenberg.
[THEME MUSIC]