On-Call Rotations with Andrew Widdowson (APW)

Andrew Widdowson (APW) shares strategies for successful on-call rotations.

MP: Hello, and welcome to Episode 7 of the Google SRE Podcast, or as we affectionately refer to it, the Prodcast. I'm your host for today, MP. And here with me is Viv.

Viv: Hi.

MP: Here with us today we have someone who I think the vast majority of SREs at Google have had the opportunity to either hear speak or have received teaching from him, and generally a really well-known name around SRE at Google. And he's here today to talk to us about on-call. Andrew, why don't you go ahead and introduce yourself?

Andrew: Hey, thanks so much for the warm welcome. I'm Andrew Widdowson, and most people at Google know me by my username, which is APW. So feel free to call me Andrew or APW. I've been an SRE at Google for coming up on 15 years now. Wow. I've enjoyed the entire time. I spent over a decade working on our Google search systems at all front-end back-end infrastructure— you name it, I had looked at it. Then I spent a considerable amount of time, about four, maybe five years working on SRE education through the SRE EDU program. And these days I sling a bunch of data in a slightly less than typical SRE context. I work on the global resilience of our physical Google offices and a bunch of data analysis about how our employees are doing. So there you have it. It's been a wild ride, but

I'm glad to talk to you today.

MP: Yeah. So on-call is something I think a lot of people in operations systems administration are already familiar with. Why would you say it's SRE's job? It's always a little bit of a funny thing that we're usually the point people for these emergencies for these huge complicated systems, but we're also not usually the experts in how any particular piece of these systems works.

Andrew: Yeah. Well, let me set the stage a little bit. So Google is a massively large company these days, but it is always been the case at Google that there have been more software engineers in a general sense, working on products, than there have been SREs. We are, if you will, naturally scarce. So while I agree that in many cases, for some of our most public-facing, high revenue, et cetera, high-risk sorts of products, that SRE should be on-call or co-on-call for a service.

The other part that I just want to mention that's a part of our leverage, a part of our scarcity, and a part of our selectivity, is that the overwhelming majority of, let's call them microservices at Google, do not have SRE on-call for them. So we pick and choose not our battles, but our responsibilities. And you can think of the fact that at least at Google, if an SRE is on-call, or an SRE team is on-call for a product, that means that there's a certain extra standard of reliability being afforded to it. But we try not to hoard that for ourselves. So even in many of those on-call rotations, we are co-on-call with our developers. Just wanted to clarify that off the top.

But why should an SRE be on-call or an SRE team be collectively on-call for a service? I think it's because we spend an extra— not necessarily disproportionate, but a necessarily proportionate— amount of our time thinking about the art and science of reliability, rather than just doing the things that make it reliable. So we are both on-call and we're thinking about how to make on-call better.

Whereas the mindshare of many other software developers who may be themselves self-on-call for a service don't necessarily have that mindshare or that amount of time allocation to do that. So, I don't know. I think we're meant to be exemplars of on-call, and we will both do a great job at on-call and like I said,

make it better.

MP: It gives you a really visceral sense of the health of your systems when you're the one carrying the pager for them.

Andrew: Absolutely. I want to be careful about how I say this because sometimes our internal phrasing can sound a little snarky, but I think it's important that our developer colleagues who don't first identify as Site Reliability Engineers— I also identify as a developer, mind you, but identify as an SRE first— our developer first counterparts need to be able to, quote, "feel the pain of the service"— the operational aspects of it— in order to be motivated to make it better, because otherwise, it's just throwing stuff over a wall. Here: take this, person who isn't really an SRE!

In my mind, SRE is about having the setup with your developers, the mutual respect to say, "We're going to operate this together. However, we may call you in for the big things, because we know that you are a calming force and that you are— especially your brain is especially predisposed to thinking through the on-call issues." So you're not in trouble if an SRE shows up. The cavalry is here. The extra help has arrived if an SRE is here.

Viv: I like it. So speaking of feeling the pain of your service going wrong, which I think you just mentioned, I think sometimes there's a perception that on-call is just really painful. It's like, "Ah, I have to carry the pager… all these things are going to be terrible. It is this burden that is part of my job" versus like, "it's a part of my job." I guess there's different ways to think about it. What, to you, makes on-call a positive if it is a positive, and how can it be a positive if it's not?

Andrew: Oh, for sure. The following might sound a little bit exaggerated, but I firmly believe it, and that is that on-call is… you get to be in the driver's seat. If you want a poor race car analogy, which I'll neglect to fully flesh out: you get to be in the driver's seat for a product for your slice of time. And so yes, if it turns out that you're— okay, bad analogies abound— if you're driving, if you're in a race car, but it's constantly backfiring, that's not going to lead to a fun lap around the racetrack. But if you're there and you're able to see the machine sing, if you're

able to get the car to really hum and you use it efficiently, you come off the track saying, "I can't wait to do that again." So for me, I look at on-call as: it's a temporary responsibility that I enter into along with everyone else who's in my on-call rotation to make sure that we have the biggest nonevent of my shift. But that being said, let's go straight into the painful on-call shifts.

So, again, the following is a cultural aspect of how Google treats its production operations. And it doesn't necessarily mean that this is right for everyone, but we have, as an homage or an honor to one of our almost founding members of the SRE org, Ben Treynor Sloss, Ben Sloss, we've named like a fatigue limit, which we've incorrectly called the Treynor limit— his last name is Sloss, we should call it the Sloss limit, whatever. We have this limit named after our grand poobah, which says: if you get more than a certain number of incidents that happen per shift, then that's a sign that your SLAs may need to be adjusted— probably do need to be adjusted.

I mean, it's not the case of, "Oh, I got paged 10 times today and that's over a threshold of, let's call it three, but this has only ever happened once. We should take radical action." No, of course not. But if you look over some smooth window of time and you realize that the amount of time you spend doing follow-through on your incidents, on your outages— which there's always a certain amount of follow-through, helping to coauthor postmortems, or seeing a fix through to production— if that is over a certain threshold, you will end up in some sort of cumulative failure, like a stack of dominoes falling over where you're like, "Oh, but I can't write a postmortem 'cause I'm getting paged again. Oh, I guess I can't write the postmortem." That sort of thing is really bad.

So we look, at least at Google, with the thresholds that we've set. And again, what even is an incident? It can be— the measuring and the math that we particularly use is less interesting than the philosophy behind it, which is the SRE or really any production-oriented on-caller has a certain mental fatigue limit that we both don't want to get up close to, but we want to very carefully defend ever getting close to sustainably, so that people can say, "I'm not depleted by my on-call shift. Instead, I'm intrigued or refreshed— best case— or nothing happened; I was bored with my

on-call shift." That's okay, too.

So anyway, that's my take on making sure that you have a balanced thing and having the managerial and cultural support to say, "This on-call rotation isn't sized for the number of incidents or the SLA that it has, or the number of people involved, et cetera." That's an important aspect of making sure that on-call is the sort of thing you don't run away from.

MP: Yeah, so something I've noticed that is not a standard across teams, but is a common practice, is to actually have both an on-call and an on-duty rotation. How does this relate to that limit?

Andrew: Sure. So just to be clear, let's define what we mean by on-call and on-duty generally. So I think on-call is: you are responsible for the vitality of the uptime, the responsiveness of the service during your shift. And on-duty means some sort of probably small quanta, but maybe high quantity of work that needs to be done: crank turning, answering tickets, answering support, whatever type of stuff.

And sometimes mechanically, some teams will make the on-caller also be the on-duty person, because there are large swaths of time where you're maybe not getting paged for your service. The goal here is not to get you exactly one page per 12 hours or something. It's to be less than— which could be zero— incidents pages, et cetera, per shift. So that's just an optimistic overloading.

But I really do want to call out that I think the skill required and the level of thinking exercised— not the level— that the type of thinking exercise for on-call and for on-duty, it could be any number of things, but we'll just say it's very different thinking. One is more like, I'm going to do a repetitive thing or I'm going to follow some decision tree repeatedly and just make a bunch of copies or whatever it is that you do when you're on-duty. But for on-call, it's much more about, as our lead of SRE Education, Jen Petoff says, being stewards of the scientific method in a pressure cooker, at least while you're being paged.

So the only thing I would guard against, if you were to ask, would be making sure

that on-duty does not drain the on-caller so that when they do get paged, say towards the end of their shift, but while they were still on-call, and they go, "Man, I just did a bajillion thousand tickets and now I'm getting paged. Oh, my head hurts." That's not setting a team up for success. So some teams, like I said, optimistically overload on-call to also include, like, "while you're here, do on-duty." But others keep it separate. And really, it comes down to how capable are you and your team of being simultaneously on-call, even if almost quote, unquote, "nothing happens", and also doing some other element of day job work.

So I particularly like coding while I'm on-call. And then I'll just make sure that I write copious amounts of structural documentation as I go. I often code comments first. So I've set myself up to be a fast— I can resume my coding quickly because I always bookmark where I was, if you will. For some people they're like, "Man, I can't get anything done when I'm on-call. So just sure, give me the on-duty." I don't know why I'm using that voice for, "Man, I'm so overloaded," but it really depends on the work. You can have on-callers who are double dutying as software engineers; you can have on-callers who are double dutying as technical program management, and so they may have different skill sets in what they do when they are not being paged.

Viv: So if you can have, say, two people splitting the work of on-call/on-duty, you're saying it could be one person, it could be two… could you have two people who are on-call for the pager at the same time? What does the people distribution look like? And are there some guidelines on setting up your rotations?

Andrew: Absolutely. So let me first answer the, "Can you split being on-call at the same time?" and then, "How would you construct maybe the demographics and distributions of people involved in an on-call rotation?" So keep me honest; let's bookmark that.

In general, I consider on-call to be both a symbol and a responsibility. So in the responsibility part, it's like, "Oh no, we're spewing 404s in our front-end or 500s or something," and you've got to go figure out why it is, and make it better, faster, more like it never happened. But it's also kind of a, like I said, a symbol— a lightning rod, if you will. So if you are the on-caller, you are being given a token

that says you are the decider for the reliability of this product. At Google, we have a saying, which is: when you're on-call for a Google product, you have temporary equivalent authority to a vice president. And that's really true. You may need to make the decision. A part of being on-call is deciding, and you may not be the most knowledged person in the decision space, but you are the most present and most stateful decider until such time as someone can augment you at your request.

So this is an indirect way of saying to your question, "What if you split the work?": so it's my personal preference to have on-call and on-duty be completely separate because I think it gives more agency to the single on-caller to say, "I choose to do whatever I do when I'm not being paged. Whatever is best for my work and day job," as opposed to I'm being told, "Please do tickets or this or that."

But if you were to split on-call by, say, having two on-callers, regardless of how on-duty is split, I think the ability to respond to an incident may be slightly improved possibly, but the authority-agency-visibility of, "Who is the on-caller? I need to talk to the on-caller" is reduced because you now have two people. You have maybe heard of phrases like, "Too many cooks in the kitchen," or, "If a problem is everyone's problem, then it is no one's problem." Or the classic, "If you don't get what you want from one parent, you ask the other."

So needless to say, I am, for the most part, in favor of [a] single person being the primary on-caller. And if you have a second person for extra support— for relieving for breaks, for "you need to go to the grocery store"— I also recommend a secondary, but exactly that: a primary and a secondary rather than two co-primaries and no secondary. That's my personal preference. I think it solves for efficacy of communication and it also solves for agency of the on-caller.

MP: Yeah. Thinking about it, the thing that would throw me off the most of trying to have two primary on-callers is, well, how you're splitting who's taking what page and doing that in some kind of equitable way. But even then, once you start, if you're just going back and forth, just alternating, there's a visibility loss there potentially— that as one page can be the consequence of the thing that

happened 15 minutes ago and isn't actually a separate problem.

Andrew: Yes. The other argument I would make here is if you were to be so unlucky as to have two problems coming in in short distance from each other, and it turns out that they are in fact completely unrelated or mostly orthogonal, having your secondary remain mentally fresh so that when you get paged again and you look at the thing you go, "Oh, that's not related to this at all. Dear secondary person, would you please take this?" Then this also reduces the cognitive burden for each of you. I think that's related to what you're saying.

MP: Yeah, definitely.

Andrew: So I think the second part of your question, if I'm hearing you right, was about how would you design parts of on-call, or would you like to throw that question back to me? I just want to make sure I understand.

Viv: Sure. Yeah, I was just asking about the rotation since we were talking about how you might staff it. Other parts of the rotation. So maybe there are more opinions on staffing, but also: how long is your rotation? What does it cover? I don't know. I know I'm throwing more questions at you in response to us bookmarking a question for later.

Andrew: This is great. I welcome all the questions. [laughs] So hear me out, fair listener, when I say the following— because again, same disclaimer as before. This is just an overview of ways that Google has chosen to solve things and size things. And so if you hear something in the following that you feel like it could never work for your company or your team, that's okay. But hear me out on some of the different trade-offs and things that we've evolved into.

So for one thing, we tend to pretty heavily staff our SRE teams. So we have an internal minimum standard of having six or maybe seven— I've lost track— people on each of two sides of an ocean, as it were. Two geographically very different sites must each have at least six or seven people in them. So that would be cumulatively 12 to 14 people at minimum in order to have an officially SRE-funded on-call team and on-call rotation. And so that has implications for all

of the rest of the math that we're about to do. You might say, "I'm a team of three, this doesn't make any sense for me!" That's totally fine. But hear me out on some of the additional timing that we do.

It is on a SRE team by SRE team— well, for that matter, any on-call team— basis to determine how much time an on-caller spends per shift, how many shifts you have per day or per week, et cetera, et cetera. But of course— and this is an important thing to flag— it is the responsibility of anyone making an on-call rotation to comply with local laws, local employment laws regarding work, work hours spent, and so forth. And so when I say the following, I mean this for a majority of teams, but there are always exceptions based on where some of our on-callers are based.

Some teams choose to have a, let's call it "seven unique days a week"— like a one day of on-call, then further split into, say, 12 hours, so like 12/12. And what this might mean is, let's say that you have different people on your team, and I'll call them A, B, C, D E. Those are all people. It might be then on day one, you have 12 hours of A and 12 hours of B. And then on the next day, you have 12 hours of C and 12 hours of D, where A and C are on one continent and B and D are on another. So that's a case of a dual-homed on-call rotation that is 12 hours within each 24 hours. So that would mean you could have conceivably up to 14 different people a week. Of course, some people may choose to coalesce and say, "I'm going to take Monday, Wednesday, Thursday," or, "I want to take Friday, Saturday, Sunday." They could choose to do that by horse-trading with their colleagues. But the idea is you roll the dice in advance and you could have a completely new person for tomorrow's 12 hours in, let's say, California time than you did in yesterday's 12 hours of California shift or whatever.

Some other teams have much more of a congealed or consecutive sort of basis. So they'll say, "Okay, well, we're still going to do 12 and 12, or maybe we move the divider between the two on-call teams because one is in a slightly different time zone where it's a little bit harder to do this. So we're going to have some sort of a mercy shift, which is like we do 10 and 14 because of whatever the case may be." Sometimes you don't control what nation your second on-call team is hired in

because it was a matter of your company staffing priorities, let's say.

So regardless of whether it's 12 and 12 in a day, or it's X and Y that sum up to 24, maybe you do formally say, "Okay, we're going to have the same person be on-call from North America during their daytime, plus or minus, for 7 days at a run." So that's much more of a different end-of-the-rails sort of setup. So you say, "I'm doing an on-call shift for 7 days, 12 hours a day. And I have a colleague who's also doing 7 and 12."

And there are also variations somewhere in the middle between these two that we've seen as well, which is, for example, not doing daily, not doing weekly, but doing something like either over the weekend plus Friday or Monday, so let's call it "Friday, Saturday, Sunday," and then having an entirely during the workweek "Tuesday, Wednesday, Thursday" setup. And by the way, I say that with a North American view on the workweek. You can imagine modifications for cultural norms in certain nations and certain countries, specifically around maybe days of Sabbath or employment law, et cetera.

But there are other variations, as well. So some people say, "We prefer on our team to know that we always have the same hours of the day that we're going to be on-call for. So give me whole days." So Friday, Saturday, Sunday was my previous example versus Monday, Tuesday, Wednesday, Thursday. Or some people say, well, actually there's a benefit to having a midday handoff just before a weekend or just after a weekend because we want to acknowledge the fact that sometimes change in our systems— in fact, very oftentimes— change in our systems, which could be reliability-impacting changes, occur due to, well, humans. And humans are diurnal creatures who are awake or asleep and that have, let's face it, very different behaviors during workdays versus maybe weekend or rest days. And so some people say, "Eh, let's do a Monday to Friday, but splitting it halfway through what would otherwise be a shift so that I get half of Monday through half of Friday and someone else gets half of Friday through half of Monday in each of our respective 12-hour shifts."

Honestly, I think the difference between a halfsies Monday-Friday split versus a Friday plus the weekend and workdays minus Friday split is minimal. I think it

may be over-optimization, but honestly, if there's a thing that resonates with your team or with your org and they would prefer to do that, give them that choice. Plus so long as you have systems that allow you to carefully trade shifts for people so that they can further micro-optimize for mutual benefit amongst pairs of people who want to do each other favors, you're going to be okay.

The last thing I'd advise for you to do, however, is just to say, "There is a robot that declares when people are on-call and it's all going to happen and you can't change, and deal with it." You have to acknowledge there are humans all throughout all of these processes. And we want to optimize for their happiness and their sustainability to want to come back to the on-call rotation.

MP: Prior to when I came to Google, I was actually part of a single site on-call rotation that had a 24-hour pager holding. I can't remember exactly how we split the weeks, but it would be multiple consecutive 24-hour periods that we'd be holding the pager for. And I'm sure there are organizations out there that don't have the ability to have dual-sided teams.

Andrew: Absolutely.

MP: So what would your recommendation for them be?

Andrew: Well, to the extent possible, I think the most important thing we need to keep in mind in any on-call design— regardless of whether it's split within a day or not, 24 hours a day or 12 hours a day— is that our humans or on-callers are our most precious resource and their freshness, their vitality— nevermind the service's vitality— is most important.

So for example, one day I came into work when I was— it was like my third or fourth week on the job. And I came in to find— and of course, this is going to sound like some Silicon valley stereotype, but please bear with me for a second— I found the engineer that I knew the most, my mentor, sleeping in a bean bag. There's the Silicon Valley stereotype, but take from it what you will. When he woke up later, I asked him, I said like, "Is that a thing that's allowed? Can we sleep in the beanbags?" And he said, "Well, I got paged late last night. And so I still

came into work today because I'm on-call a little bit later today, but I was doing the company a favor last night and staying up later to make sure that I saw this through. And I know that if I get paged, it will page me in this case [during] what would otherwise be my business hours, daytime, but I'm going to take a bit of a nap so that I can be much more fresh in case I'm not paged until later today. I want to do better at that page." So yes, "be easy on yourself" was the directive that he gave me. And I remember that to this day.

But maybe the general lesson to take away from this is, if you're going to be doing an all-day and all-night on-call rotation, I don't think it is sustainable personally to be paged multiple times in the middle of the night for multiple nights if the type of work you are doing is not shift work. I know certain classes of engineering work are like, "Oh, I'm going to roll onto the night shift and I'm going to roll off." That's a different story. But if you are a quote-unquote, "daylight hours" worker, 40 hours a week, whatever the case is, but you are also on-call, the only thing I would ask of a management structure in that is to have compassion for the fact that if people are woken up in the middle of the night multiple nights, they're not going to be at their best for later times.

So what might be a compromise here? I would maybe suggest that maybe there's a policy— like if you get paged in the middle of the night on multiple consecutive nights, that there's a mercy substitution, like you can avail of your colleagues to see if someone will take over the rest of your shift, knowing full well that most of the time, if things are sized right, crossing your fingers, this won't happen. But in the rare case it does, we let you tap the hours, is essentially what I'm saying. That would be my suggestion. But again, given that that is not how things work at Google, I'm merely speculating. And I wish everyone the best of luck in figuring out how to size, shard, and trade their on-call rotations.

Viv: I really liked that you said that the people come first.

Andrew: Yes.

Viv: I'm just using myself as a reference point, but when I first started with on-call, I was really nervous about being on the rotation. You don't want to let the

team down, you want to make sure you're getting to everything. And I think it is a good reminder that you have to make sure you are in your best spot, which ultimately will help the rotation, too.

Andrew: Exactly. It's not like we're like, "Welcome to SRE. We're going to burn your amygdala out from stressing you out for multiple days on end. And you're going to start smelling colors." That's not what we're here for. It's the mythos, the kind of the reputation that we put around SRE at the risk of over-mythologizing SRE, which is itself dangerous.

But what we generally tell people when they join the SRE org at Google is, "Welcome to this specialty role. It's going to require you to grow as an engineer. If you are a software engineer, now you need to learn some more operational things. If you were more operational-oriented, you're going to learn more software things. You will become a hybrid role. Not only will that make you a better engineer, we believe, but also it means that we will use you more selectively for the better good of Google products. And so with that, we are going to make major investments in you and in your operational, as well as psychological, safety. Obviously, we'll treat you well in all other respects, as well. But not only will it be you, you won't be alone. You'll be with a team of people who will be your fellow on-callers and people who are responsible for making the reliability of your product better."

Because of this— it's essentially, everyone loves being told in some way, "You're special." But we try to embody that in what we bring people into, because we know that there will also be times that are very challenging and hard, and "I don't know the answer" and "do I look bad?" and so forth. And so we try to empower people to know, "You are the best person for the job at the time. Decisions will be made and systems will become more stable, but we're not going to burn you out or leave you on a ledge because of it."

MP: Digging more into the staffing question, when it's time to turn up a new on-call rotation, to have a… I guess, in the Google structure, you would say that dev— the engineering organization— has decided that they think it is worthwhile, that it's worth the investment to have SREs for this part of the system. And there

now needs to be a new on-call rotation that didn't previously exist before. How do you get from that state— where really the developers are the keepers of all the knowledge and all the information about how the system works— to also having this SRE team alongside them that is ready to respond in an emergency?

Andrew: Yes. So there are a lot of different ways to barn raise or bootstrap, if you will, an on-call rotation for a product. And especially if you're going from an existing on-call rotation that was self on-call by the developers of the product and there were no SREs involved, to going to one where the developers may still occasionally be in the on-call rotation, but the overwhelming majority of time is attended to by SREs— there are a couple of different ways you can do this that I think will lead to a more sustainable, less painful, better results, faster sort of a thing.

So allow me to paint a picture of maybe an over-design of things you could do. Pick from this— any of these recipes. And I think it will be better than just turning the lights on an on-call rotation and throwing the pager over a wall.

So perhaps with the talent pool that you have, maybe you already have a mature set of SRE team or teams, and you have identified the most senior folks within those teams, but they're on-call for completely different products at your company. Like we like to say, titrate onto any new experiences. That's not only pro advice in the mental health space; it's like saying, "don't put in the hot liquid or you'll curdle the milk" kind of a thing. It's, if you can, try to seed a new SRE engagement with one or more senior, already experienced SREs, even if they are not in the same product's domain. Hypothetically, let's say we were turning on Google Maps on-call for the first time. So what if the person you bring over was in Search? They've had experience with on-call and reliability. So if you have that, I would start with that.

The next most important suggestion I would have is: there was a presumption here that you do not have an infinite supply of fully grizzled veteran SREs or whatever it is— fully experienced SREs to staff all of your new on-call rotations. And that eventually, if not as your second or third hire, certainly as your fourth or sixth hire, you may have someone who's completely new to the SRE role, but you

hire them because they're an intelligent thinker— they're an engineer in the making, or they're an engineer in their experience, but they're an SRE in the making.

When you get to the point where you have people who are both learning the product and the role of SRE simultaneously, it is vital, in my opinion, that you have a very good safety net under and around them. What I mean by this is education and being able to walk the walk and being able to posture yourself as an SRE. Being able to emulate the thinking— the scientific thinking that goes into being an SRE— all of that's important. And the last thing you want to do is raise someone in a vacuum. I know we overly use the phrase "cargo cult engineering" in engineering. I think it's an unfortunate phrase, but many of you know what I mean: we don't want people to reinvent SRE in a vacuum 12 hours in the future from their other on-call colleagues. Because at the end of the day, thinking about this from a psychological standpoint, SREs are engineers who are already rapid inference-makers. Engineers, you know, trying to measure a couple of times, cut once or twice, make a better bridge, whatever it is that engineers do, but SREs specifically tend to be hired because they are in the good sense of bias; they are rapid, biased decision makers. And so this doesn't just apply to how they solve that we're serving excessive 500s in our front end. It's not just technical challenges. They're inherently biased creatures when it comes to things like, "Do we trust our fellow on-callers? I can't detach from work because we have someone junior, and what if they get paged and we still need to keep the site running?"

So this is a very long way of saying: I think it's very important that you have fully participatory education of all existing team members, to raise the next set of SREs that you hire and bring into the team, for everyone's benefit. Not only will it accelerate the journey to on-call for the newbie, as it were, but it'll also increase trust amongst the rest of the team.

Ideally, you don't have an overly judgmental, clique-ish, middle school-style SRE team. "No, you can't sit at our lunch table," nothing like that, certainly. But you want people to be able to see that your newest people succeed or fail at a theoretical exercise multiple times before they get their hands on the control

surface and before they are solo on-call.

So when I say this, my recipe, then, for an SRE on-call team is clear. The first person should be highly experienced. And the point that I made through that long-winded explanation is: the second person should be someone who is relatively senior but is incredibly good at helping teach the art to anyone else and helping to further stir the cultural pot of norms going on in your team.

And then the third plus folks can either be senior talent if you've got it, or junior talent if you want to build it. But any way you slice it, you need to make sure that you have people who are good at norm-ing the team as you go. There are other, of course, considerations at play as well, including: when do you hire? Do you start with a 24-hour rotation or do you start with the 12 and 12, plus or minus? How do you make sure that both— assuming that you're dual-homing your on-call rotation— both sites feel equally empowered, and it's not that one is domineering over the other, or that only decisions are made in the same time zone as where your developers live, or something like this. There's a lot of that.

But on the whole, in summary, in barn raising an on-call rotation, I look for super-senior— like exemplars of the art— then teachers of the arts— which can often also be exemplars of the arts and ideally are— and then the healthy mix of people from different sorts of backgrounds within all of this. And again, I just want to encourage you to say that it's okay if an SRE comes from a radically different product within the company or elsewhere into an SRE team. If anything, as our biologists would say, this adds hybrid vigor. So it's great if you've got a front-end-oriented person and a throughput or pipeline-oriented person working in the same SRE team. Oh, and by the way, that new SRE team is a storage layer that isn't front-end or throughput. That's actually one of the best cases for eventual awesomeness: robustness of awesomeness.

MP: Could you go into a little more detail about creating an opportunity for everyone on the team to build their own confidence in their teammates' abilities?

Andrew: Yes. And failing in a theoretical exercise rather than on the real product.

Yes.

MP: Yeah. Can you talk a bit more about what that would look like for a team?

Andrew: This is one of actually my favorite things to talk about, so I'm so glad that you led us here. Part of being in this tribe, if you will, of SRE is the kind of oral culture. Part of this is the lessons you learned from your predecessors and then eventually your colleagues. And so one of the best ways to have really good muscle memory at diagnosing problems, becoming better at applying the scientific method as it were in your on-call, is to practice it in a no or low-stakes environment ahead of time.

In some circles— especially gaming circles— you might hear of tabletop-style exercises with game masters who are running you through a multi-player adventure. You're in a twisted maze of 500 error codes and Prometheus consoles. What do you do next? I go left. Okay. What do you see there? Like if you have your own multi-user game, that's a lovely environment. Other security professionals will also use the word tabletop exercise. It's like, we're going to see what happens if we have a certain country invade another country. What do we do? Who do you call? That kind of a thing.

At Google, we have what we call "Wheel of Misfortune," which is just a cheesy way of saying that we do a tabletop. And what we do is: many teams will have a weekly or monthly sort of gathering. They'll call it Wheel of Misfortune or something else. It's a tabletop where the game master has been thinking, since the last game anyway, about an interesting story, an interesting diagnostic path that they want to socially norm out to the rest of their team. They want to inoculate them to the, "Well, I've never thought of this before" problem.

So they'll come up with some sort of trigger condition and some sort of actions that need to occur in order to save the universe or save your product from certain ruin. So maybe it'll be that the whole team gets together in a big room with a whiteboard or a computer running a synthetic copy of your services stack, and they'll say, "Okay, well, today's volunteer is Sally. And okay, Sally, welcome to being our brave on-caller for today. Your backup on-caller today is me. You get paged

because of the following message." And it's something authentic to how your alerting and monitoring systems work at your company. And maybe you don't have screenshots of a real thing that's broken or maybe you do; maybe it's all verbal. So the person says, "Okay, Sally, so you get paged for excessive 500s in Asia Pacific. Where would you go to validate this? What would you click on? What would you see?" And maybe Sally says, "Okay, well, so I open up our Prometheus dashboard and I filter it out so that I'm only looking at the Asia Pacific region. And then I'm looking at what cloud clusters we're running from. What do I see, Andrew?" And as the game master, I would say, "Okay, well, you see a bunch of what looks like very cyclical traffic going over the last eight days. Latency seems within normal, but there's— oh, there's this little spike over here in the very specific cluster that we've rented in India." "Oh, tell me more." And so it becomes a conversation.

Now meanwhile, everyone else in the room is listening into this. And I think there are very likely different sorts of cognitive processes going on. So for example, a very senior person is going to be like, "Oh, Andrew has given them the good old 500s error and APAC trick. I bet this is from an issue that we had three years ago, and he's just doing this as a rerun." But maybe there's a junior person in there and they're going, "Boy, I'm not thinking about this the same way Sally was thinking about this. That's cool. I've learned something from where Sally is going with this"— that sort of a thing.

Now keep in mind that the volunteer— this mythical Sally person that we're talking about— it's okay if they get stuck, because also that goes back to the whole norm-ing of the tribe. If you don't know an answer, that's okay. This is not like, "Okay, now please go commit ritual embarrassment, and be shunned from the team." Instead it's, "Okay, we'll phone a friend. Does anyone else here have a suggestion?" And the game master is very good about making sure that they're not just always getting the smarty-pants person who's been here for 10 years answering the questions. And you just stir this rigorous scientific thinking through your team.

Bonus points if occasionally your game masters do something forward-looking. For example, the team is going to be on-call for a new feature in, okay, if this is

the on-call for [the] Maps team, there's a new feature coming down the road where you'll be able to get directions by skateboard instead of by walking or biking, let's say. So maybe one person on the team has been working with the developers on the skateboard navigation feature. And so they're the game master that week. And they say, "The following scenario happens three months in the future when we're on-call for the skateboard backend." And people go, "Well, I don't know anything about the skateboard backend." "True. Let's learn about it together."

So it really is a great way to build mutual trust, muscle memory, better neurons for rapid inference in the myriad decision tree that you'll be faced with when you're on-call, and some level of hope/strategy for the future. I love it and I recommend it for anyone. You don't have to have played a bunch of role-playing games as a kid to become great at either participating in or proctoring a tabletop. You just have to have some interest in it, some willingness to be a bit verbal— a bit more communicating than you might be in your day-to-day sort of job— and be fascinated with how other people learn or how you want to learn.

Viv: Absolutely. I also really like that. I'd just like to point out that at least for Google, we have SRE EDU, which does do some form of this. And that's, I guess, the training program for new SREs. And so you get this upfront of like, "Oh, let's test out the waters."

Andrew: Exactly. I co-founded that group, and that was very, very important in our design. We wanted to make sure that people got a front-row seat to theoretical fireworks so that they didn't have to be scared the first time— like, "Oh no, I'm letting Google down." Without giving too much of the secret sauce away, I will say that a really cool thing about being an SRE at Google is that in your second or third week on the job, you get to be on-call for a hypothetical Google service: fully featured, but [that] doesn't have any real users. It's just robots pretending to send clicks.

And you learn both about how technologies at Google work and how they break. And then, "Oh, huh. Why is this pager on my table going off?" And before you know it, if you were given a lecture about internal corporate networking or

production networking at Google, you're getting paged because of something that ultimately requires you to fix a part of the network that your stack runs on. And it's really cool. So by the end of the week, you've saved the universe four or five times. Of little consequence because that universe is synthetic, but it's a real conversation starter and it's a real memory jogger.

So what can you take away from this as maybe a listener of the Prodcast? That is, semi-realistic but low stakes opportunities for your newest or your newest to have been transferred onto a team within your company— opportunities for folks to test the waters out without becoming shamed or embarrassed— are so key. And it shouldn't just be at the beginning. Ideally, it should be something that happens regularly for everyone, because then you've incentivized learning and getting better as a group, as opposed to, "Well, so-and-so's a screw up." It goes to the whole culture of blamelessness that we championed as being SREs.

MP: Well, thank you so much, Andrew. It has been an absolute pleasure talking with you today. It's been great to hear all of the knowledge and the insight that you have in this domain. I definitely don't think I'll think about on-call rotations quite the same after this conversation.

Andrew: Well, I'm really glad that y'all invited me on. I think it's wonderful that the Prodcast is now reaching larger audiences, and let's keep the conversation going.

If people have any questions about what we've talked about today, come find me on LinkedIn or something like that and I'll be happy to answer some more questions. But it's fascinating to me to see how the art and science of SRE has evolved both inside of Google and throughout the whole world. This is not a cliche when I say that I'm really excited to see what happens next. I'll be there, and I'll try to be refreshed enough to enjoy it, as I think we all should. So thanks again. Take care.

Viv: Thank you.

MP: Take care.