Google Prodcast Season Three Episode

STEVE: Welcome to season three of the Prodcast, Google's podcast about site reliability, engineering and production software. I'm your host, Steve McGhee.

This season, we're going to focus on designing and building software in SREs. Our guests come from a variety of roles, both inside and outside of Google. Happy listening. And remember, hope is not a strategy.

[JAVI BELTRAN, "TELEBOT"]

Hello everyone, and welcome to Google's SRE podcast, or as we like to call it, the Prodcast. This is the first episode, or as we like to say, the zeroth episode, in season three. Today, we have Vlad and Amy, who are going to help us out a little bit.

And we're going to start off our season. This is going to be the third season. We're going to be focusing on software engineering in SRE. And the topic we're going to focus on today is going to be what happens when software engineers solve production problems? And what's the importance of the software engineering mindset within SRE?

And this in no way means like software engineering is the only thing, or like you should only be writing code 100% of the time. Certainly not. In fact, I think we'll get into that a bit. But before we do that, why don't we have our guests introduce themselves?

So let's go alphabetically because that's the best. Everyone likes it. And Amy.

AMY: Yeah, hi, folks. I'm Amy Tobey. I have been doing tech in what we call SRE for about 25 years. So before that name came around, that's what I was doing very quickly out of college. Today, I work at Equinix as a senior principal engineer where I meddle in product work and SRE work and software work-- just all over the place. Awesome.

STEVE: Thanks, Amy. And Vlad.

VLAD: Hi Hello. Thanks for having me on the podcast, or on the Prodcast.

[CHUCKLES]

And my name is Vlad. And I work for a big company called Siemens Healthineers, which is the health care arm of Siemens. And there, I'm responsible for the so-called Teamplay digital health platform, which is a software as a service platform for health care applications and digital services.

And this is where SRE comes in very handy because you need to operate the platform somehow. And this is where we use heavily the SRE methodology. And I'm looking forward to talking about it with all of you on the podcast.

STEVE: Well, thank you both for coming today. Just to get started, I think we want to start actually with the inverse. So specifically, we have a question here for Vlad, which was, how do operations serve as the core part of SRE, which might feed into software engineering later on? Like, what do we even mean by operations to begin with.

VLAD: So in my view, **SRE is a methodology for running digital services reliably at scale. And it's about running the services, so we are firmly in the operations arena. But the curious thing about SRE is that it applies a lot of software engineering methodologies in order to achieve that-- reliable operations at scale.**

So it requires you, for example, to implement some sort of SRE infrastructure that enables development teams to operate their services with acceptable cognitive load. And it then requires the software engineers to actually use that infrastructure in order to do the operations of the services. So it requires the operations folks to implement something as software engineers to be used by software engineers, and then requires the software engineers to go on call for their services for an agreed period of time of the day, so working as operations engineers. So I think this is what makes it interesting, and this is where we see the software engineering making its way into SRE. So that's on the one hand, I'd say.

But also on the other hand, if you take a broader view of what SRE is trying to achieve, it's then actually reliable and scalable services. And that means you need to weave in that SRE thinking from the beginning to end of the service life cycle. So that means you need to apply software engineering things, like what kinds of stability patterns do we implement now into these services?

Where do we put circuit breakers? Where do we need bulkheads and so on? So you need to have some solid background in software engineering in order to do this across the life cycle.

STEVE: Totally agree. Amy, how do you feel about Vlad's definition so far.

AMY: I like most of it. I tend to take a little bit more expansive and sideways view of it, because what I've seen a lot of in the field is a lot of people look at SRE, and they're overemphasized on the software part. And then we talk to our friends at Google, and they're like, no, no. It's really about what a software engineer would do in an operational context.

And I think they're both wrong, because like when I see effective SRE that I've done out in the real world-- and I'm just going to say Google's kind of not like the real world. It's this magical fairy place where people get to have fun and totally have cool tools and good software tools. It's a real place.

But what I'm getting at is, the real essence of what we were working on are these feedback cycles across the software engineering life cycle. So starting at the very beginning, we want to get involved in product management and in product conception, and start to insert reliability and security-- because it's always on the way for me, requirements into the product process. And then all the way at the end, if there is an end-- well, there is an end when we actually shut the service down. But maybe that long middle, where we spend most of our time, we're constantly looking at where can we improve the feedback signals. And we implement that through a bunch of different ways, like SLOs going on call, all of the DevOps kind of stuff that we try to integrate, CI/CD, it really comes down to how I tighten those feedback cycles between what my customer is experiencing, what my software engineers are doing, and what my leadership believes we're all doing.

STEVE: Yeah, I think that last bit is actually a great one as well, which is, what does the leadership think we're doing. One take on this historically has been the idea of not accepting 100% reliability as even a reasonable goal. The jokey way I like to describe it is, if you're asking for something that's unreasonable, you're just telling people to lie to you. And this is what happens with teams that give unachievable goals like 100% availability or reliability in some way.

AMY: It's true. You look at 100% stat, and you peer at it long enough and you stare through it and go look at the actual metrics, somewhere some statistical significance is being dropped.

STEVE: Yeah, or it's 100%, except for x, y, and z because we're leaving those out, because come on. you wouldn't want to leave those in.

AMY: Because they're just not convenient.

STEVE: Yeah, that's right. Have either of you encountered a situation where something like this is going on, where the metrics look great? And when you dig into them further, it turns out something funny is going on?

If that has happened, what do you do? What's the thing here. Is it just talking to the teams in general, or talking to the leadership teams and saying, like, hey, don't do that? Or is there something more subtle here?

VLAD: Well, I think that happens all the time with the definition of SLO. So you go to a team and they're totally new to the party. So they've never done any SREs. And you start advising them, OK, so maybe we would select some SLIs that would apply to your service.

And then inevitably, they come up with something like availability. That's important for any service. And then you work with them, and then they come up with an initial set of SLOs.

And they think, OK, so our service is so and so and so available. And then what happens is, well, you feed all this into the SRE infrastructure. And then a couple of days later, everything is red because all the SLOs have been broken.

And you come to them and say, OK, now have you looked at the SLOs, first of all? No, because they're not trained yet to do this. But then you show them, OK, so all the SLOs have been broken. What does that mean? Well, actually, the customer hasn't called up and didn't say that something is wrong.

So there, you've got already the cognitive dissonance. So the SLOs have been defined. They are all red. But the customer actually didn't experience it as being so painful. And that happens all the time until you get the team to a point where those things converge.

So the definition converges with the user experience. And when it's red, then it actually means the user experience got broken. And when it's green, it actually means the user experience is OK. And once you get them to a point that this is an ongoing refinement process, then this is where you know, OK, you are getting it. You're getting somewhere.

STEVE: Totally

AMY: I have run into that. In fact, there's still a dashboard I'm working slowly to eliminate that shows a number of services-- SLOs that is measured by an instrument that I don't particularly find very reliable because it doesn't represent the customer experience. And there's no technical problem here. It's actually really interesting about it.

Like technically, the system works as designed. It is instrumenting as designed. It is reporting metrics that are actually somewhat representative of what the system is doing, but have almost no bearing on the customer experience. That's the first problem.

But the way I approach this is not to go and start poking at the technical stuff. I tried that for years. And basically, every time I poke at it, they're like, well, everything's working exactly as we designed it. Well, yeah, because you designed the wrong thing.

But the thing is, it's been in place for years. And so this is my world. It's been in place for a long time.

And so every time I try to approach it and be like, I think we could do this better, everybody's like, well, no. It's been there the whole time I've worked here. It's fine. So this is really more where we start to get really deep into leadership and having to go around to the product managers, and not just the people who own, like, the monitoring system there, but each of the product teams' product managers and point out to them how broken this is. And then go back through their incidents and go like this one, this one, and this one and this one and this one and this one, all caught you by surprise because you're monitoring doesn't actually work, even

though you go look at the dashboard.

But still, it would be like you would show somebody this fairly obvious, to us, truth. And they would go, wow, I got to fix that. Except there are so many other things going on that largely, they're just like, OK, yeah, that's really important. But I've got to go work on this other much more important thing.

And so it's still there, and we're still chipping away at it. And we've got most of those people starting to do higher level monitoring so that they come through this journey where, instead of me telling them that it's wrong, basically, we have to take them down this longer road where through, like, instrumenting with OpenTelemetry and using more modern tools, eventually, these folks will come around and go, yeah, that's really bad. And then they'll tear it down. But we're just not there yet.

STEVE: This reminds me of like-- this is a high-level, hand-wavy definition, but the difference between programming and, like, professional product software engineering. You can technically do programming just fine and it'll do what you say. But if you're not designing the right thing at the right time for the right people, and getting feedback and doing the whole thing, and working with your peers and like collaborating, like there's a bigger, bigger thing over there, which is capital S, capital E software engineering. We put a W in there sometimes for some reason. It's funny.

AMY: I tell people all the time, I write code for free. I get paid to do Jira.

STEVE: So there's a trend going around these days around platform engineering. And sometimes, we talk about internal product management. One way that I've heard SREs described is just like another set of influences, or influencers maybe, to that internal product or to that platform in terms of representing the reliability or the security of the things that are being shipped through that internal platform. Do you guys have any takes on that or experience in thinking about reliability through this internal platform concept? Or does it just happen that way?

VLAD: Well, I would say that the SRE infrastructure that you're building ideally would feed into the bigger platform that you designed for your engineering organization. It would feed naturally into what you've got. So it might start as a separate thing.

But overall, it'll be great to get to a point where the platform that allows you to do pipelines also allows you to set SLOs and feed them into the infrastructure and getting the bridges reported to you, all in one more or less seamless experience from the engineering point of view. I think this is kind of a place where you need to get to. And I think in order to get there, you need to have product thinking for the platform. And that means you need to have product owners-- you might call them technical product owners for the thing. Otherwise, there is no end of product thinking there. And you are not going to end up having a product [INAUDIBLE]

probably a bunch of stuff that is engineering-led, but doesn't look like a product.

STEVE: Totally.

AMY: I think platform engineering is a little bit of a mistake, and it's all Google's fault. And it's because one of the things that they left out of the Google book, or the original SRE book, that I think drives everybody outside of Google crazy-- especially me, is the leadership structure that's in place at Google that enables SREs to speak up and be heard. So this is something I've learned over the years from various folks I've known that have worked there.

But there are very high-level leaders-- I think all the way up to the senior vice president level, who represent reliability. And most businesses have that for security, but they don't have it for reliability. That's pretty unique to Google and maybe a few other places.

And so I think a lot of why platform engineering is arising is because, well, one, it lets us take a lot of these concerns that we're building and actually build a platform where we can just encode them into the platform. So the developers don't even really have to think about it. But the reason that we're driven to do it that way is because without that power base-- I'm going to talk about just power dynamics here.

Without some kind of platform, with some kind of buy-in and gatekeeping-- really, just gatekeeping, it is super-duper hard to be heard as an SRE because you're just this voice out in the wild. And like I was saying before, as most of the leadership that are trying to ship products and keep the business going and make their P&L go up and to the right, they're just not thinking about it. And they don't want to hear about it. They really don't.

They don't want me showing up on their door and going, hey, buddy, you got a huge architectural risk right there. And that thing is going to catch on fire one of these days. You're going to have a bad time. I could fix it now. It's going to cost you half a million bucks.

And they go, I'm just going to roll a dice, because I don't have any power. I can show them the risk and they can make a business decision on that risk. But humans are notoriously bad at this. And so I think that's really where platform engineering is truly an evolution of SRE and DevOps and all of the things that came before it. But the real pivot that it's bringing to the table is that installation of somebody in power who has a way to say to the leadership team, like, no, you're actually going to do reliability, and that's not an option.

STEVE: Yeah, I mean, part of modern-- or not even necessarily modern,ut, as we were saying, professional software engineering is prioritization, or being able to say, like, we're going to build this thing and not that thing. And when priority inversion happens, it's often because of a lack of information or because of a lack of sometimes a champion. So having a seat at the table during that prioritization is super-important.

And I agree. That was completely missing from the book. I think that was even left out on purpose because we didn't want to create any "you have to be like Google" vibes. But maybe at least just saying it would have been helpful in that structure-- in that book that people saw. Let's keep going.

So, Amy, how about when you have operators and sysadmins that haven't done software engineering in the past? Have you had any good experiences in like leveling them up and training and all these kinds of things? Like, a, does it work, b, how difficult is it-- c, d, e, up to u. How do you feel about this?

AMY: Yeah, yeah, yeah. So A, it does work. B, it's super-duper hard. It's the hardest problem in business. Like, how do I up-level a workforce. In any domain, go and ask some business leader, what's the hardest problem for them. It's like, how do I uplevel my workforce?

So just out of the gate, the typical move that we've all seen-- and I'll just repeat it, so we're all on the same page. A lot of places looked at the Google book. They probably didn't read it, to be honest. They flipped through it and they went, look, this might help me. I like reliability.

And they renamed their sysadmin team to SRE-- all the story since SRE was invented. And when that happens, it has this effect. It does actually do something. So to sit here and say renaming the team does nothing, that's a lie because we've reoriented their focus a little bit right out of the gate.

So you start to see teams-- I've seen this probably four or five times, where some team gets flipped to SRE. And they're largely just an operations team, and the good ones are doing Kubernetes and they're starting to iterate towards a platform. But they don't have that empowerment. And so that's the big gap, I think.

There's two major things. There's a skill gap, but there's also the empowerment gap. And then we end up in a livelock with the business, because the business is like, well, I'm not going to trust you with this responsibility because you don't have the skills.

And then the people sitting in the seats are like, well, how would I go learn that if I'm not trusted with it? And so we just go around, and so they get stuck doing operational work. And there's all kinds of classic goodies in operational work where it is so hard to get your head above water to where you can actually start swimming.

So you just end up bobbing up and down and keeping your face above water. That's most of operations, most places. So you have to start with incremental growth in the SRE and bringing your people along.

And that's the only thing I've seen that works. You can do the flip, but you have to start to peel off some of the roil and do it incrementally so that you make enough room to start

implementing one SLO. And then you make enough room to build another SLO or an incident management, or do a retro program or something. And just snowball that.

But I've never actually seen anybody flip it, and then everybody goes, yes, I'm an SRE now. I'm going to go learn how to be a great software engineer. They go, like, what the hell is a sprint? And why would I sprint for two weeks?

That's what most people say, and usually where we start is with those processes. So yes, you can do it. It takes investment and it takes empowerment.

VLAD: And I know a case where the team wasn't just renamed into SRE, but they got a raise as well. And nothing else changed. The leadership thought, OK, if you raise the salaries, if you rename the team, then they will live up to the challenge. But on the ground, nothing changed.

So what I found useful is, if you have got an operations team that has done traditional operations all along, and you inject a real software engineer into the team-- so just a software engineer that's got an interest in infrastructure stuff and automation and so on. Then this can actually catalyze a change where the software engineer starts building something as a software engineer for a certain purpose for a certain user and so on. This is where things can catch on, because then they go to lunch together and so on.

They have conversations on a daily basis. And they've got a connection then to a software engineer which is so close, they have never had that before. And this is where you can catalyze the change.

AMY: I love that. So embedded SRE is wrong. Embedded SWE is right.

[CHUCKLES]

I'm not going to say either one is right or wrong. I just thought that would be funny.

STEVE: That was the missing chapter in the book. We fixed it. Good job everyone.

One of the jobs or one of the purposes of SRE in a group is scaling. It's like the system is already there. Sure, it's going to be adapted over time. But one of the challenges is like just handling growth and stuff.

Stuff happens at certain inflection points along the way. Do you have any stories or insights around pitfalls, or the opposite of pitfalls, whatever those are, around scaling of these types of systems, whether they're operational or complete re-architecting? How do you see them coming, how do you approach them, et cetera?

VLAD: I think something like this could be a great catalyst to actually roll out a proper process

for operations, like SRE for example. Because if you start to scale-- so the number of users is growing, the session length is growing, and so on, then that will place higher operational demand onto your services, which means they will need to be more available than before, when the load was smaller. But then if you don't have the processes and the cultural substrate in order to support that availability, then this is where you will come to a point where you understand, OK, we need to make some drastic change here.

Otherwise, we are not going to be able to handle it. So we'll just not be able to provide the availability that's needed for that type of traffic that we have never seen. So I think this could actually lead to a great crisis that you can use in order to start introducing proper processes.

AMY: That's how it often goes. Like, you wait for an incident. And then that creates the lightning in a bottle, and then you can point that at a team and empower them to go and fix the problem. So I think Steve asked about, like, how do you predict it.

But this is ultimately how most businesses get it done, really. I think there's a rare privilege a few of us have had-- like rarely in our careers even, where we're somewhere where things are like, we can see the next 10X coming, we can see the 100X coming. Usually what happens, though, is we join a business in a somewhat steady state where it's at some single-digit growth rate.

And then the actual problem we have isn't scaling up. So I think most SREs-- most folks carrying that title out there, have the opposite problem, which is how do we make things scale down. Because almost nothing is designed to scale down. Lots of things scale up just fine.

Kubernetes scales up like a dream. Amazon scales up like a dream. You can add more instances. Web servers scale like they're super easy.

I don't say super easy, but straightforward. We have good tools we can apply to the problem. But in a lot of enterprises, when we talk about scale, what we actually run into is things that don't scale down. And so we have either like programs that want to be resident in memory 24/7 so that they have their own little built-in schedulers and stuff.

And so they're all tightly coupled codes. Like, yep, you're stuck spending n thousand dollars a month to keep n servers up to keep those things running because they can't scale down. And so that has a real business cost.

So when I think of scale, that's most of the problem that I've worked on in the last few years. It isn't the other direction because that's usually like, add more Cassandra nodes, add more Kubernetes nodes, we're good. But instead, how do I build something small and economical, and then iterate on it while it scales up? And that's actually, I think, far more interesting for the majority of folks out there doing enterprise work.

STEVE: It turns out cost matters. Funny, huh?

VLAD: Indeed.

[CHUCKLES]

AMY: It just depends on if you're like directly in a business that has AI in its name or not.

STEVE: All snark is welcome at all times here, by the way. Speaking of which, do you have any examples of a time where a tool, method, product, thing just changed the way that you or your team looked at or worked in production, either for better or for worse? Like, there was this thing that changed the way we all approached production as a whole, whether it was a way of thinking or a piece of technology or someone's way of thinking. I don't know, something like that.

VLAD: So for me, it was clearly getting the development teams that had never done operations before to actually look into production by applying things that SRE suggests by defining the SLOs, thinking about the journeys that would deserve SLOs, then feeding the SLOs into this infrastructure, starting getting feedback from this infrastructure. And then the crux comes, reacting to the feedback from production, enacting that powerful feedback loop of defining something. Then putting it into production, getting feedback from it, and then reacting again, instantiating the scientific method in the operations arena, so to speak. I mean, this just completely changes the attitude of software engineers to what they build, because they suddenly don't just build it. But they build it to observe it.

AMY: I 100% agree with Vlad on that. I might add a bit of color on around the last thing that he said, which was observability. I think today, we're in a new world of observability than we were even five years ago.

Just with the OpenTelemetry out there, instrumentation for a long time out in the world, we had monitoring vendors who had SDKs that you could hook up to your code. And it would auto-instrument it and give you this wealth of dashboards for free. There's some of them very famous for that. I'm not going to mention their names because they sometimes get upset with me.

But that's shifted. It's all open source now. OpenTelemetry has re-implemented most of that auto-telemetry, so there's not much excuse not to monitor your systems anymore, because largely what you do is point OpenTelemetry at it and it auto-instruments most of your network calls. So you get an 80% solution for fairly cheap.

And then there's this wealth of tools that have evolved over the last few years, largely out of inspiration from the hyperscalers. Like Facebook Scuba evolved into Honeycomb, which is something that I've been using in my work in the last few years with OpenTelemetry. And that's

enabled that cycle that Vlad described. It's a small part of it. But part of that cycle is our developers, we can put them in the production stream and that starts to create empathy with the operations.

But the observability cycle is what starts to change their mental models such that they start to design for operations earlier in the process. So that's where I think the modern observability, where we have distributed tracing, just is so easy now compared to even just a few years back. It starts to change the game because now I can roll it out and see what is it actually doing, as opposed to, like, do these numbers on this graph look kind of what I imagined they should look like from my mental model of the system? It's a very different world now. So I think, yeah, tools like Lightstep and Honeycomb, and some of the modern, like Grafana and stuff, is starting to really get there and make this faster.

STEVE: This might be leading the witness a little bit, but that's OK. I'm curious, when you introduce a system like this, and you introduce just conceptually-- like, hey, we can take this X-ray to production, we can understand its bones at a pretty deep way. Does that land pretty quickly with developer teams who have never done it before. This is the leading part.

AMY: It doesn't.

STEVE: But how do you fix that? Like, how do you get it to work?

AMY: The way I did it at Equinix was, I embarked on, like, a six-month project where I just would stop by repositories, add the OpenTelemetry instrumentation, work with the team, get it deployed, get it turned on, and pushing out to the metric system. And I really didn't get a lot of uptake until I got almost all the way to the end when I could start to show developers and be like, hey, when you do this with our API, these are all the things that happen in here. And then they start going, what the heck is that? That thing pulls? Why is it polling?

And why is it polling every 10 milliseconds. Like, what the heck? That's where you want to get them.

But until you can show them a waterfall of most of the system, it doesn't click, is what I found. There's a few people who, like, get it. You say, like, it's a directed acyclic graph.

And they go, oh, I love this. Let me go do some tracing. And everybody else has to see it. And then they're like, that's pretty cool. And then they start working with it.

VLAD: Yeah, my experience is the same. It takes a long time to get a development team that has never done operations before to a point where they really care, to a point where they notice you and so on. And that requires team-based coaching, I found ideally, where you really get the people together, including the product owner, who is supposed to own the service end to end.

And then really work with them and getting them to a point where there is evidence where you start having insights like seeing is believing. OK, I've seen this, so now I believe it, and so on. It definitely requires a lot of dedication and coaching. And also, once you think that you've got them to a point where they do this, come back to them half a year later and check whether they still do it or not.

AMY: Yeah, that's another good one.

VLAD: Yeah, and another interesting bit is, we mentioned before the cost consciousness in the cloud. That can be a good driver for this as well, because if suddenly your login costs skyrocket and only your CFO finds that out, then this is definitely already too late. Because then you are accruing the cost at a much faster rate than you should. And if that gets pushed onto the organization to actually reduce costs, then you can only reduce costs once you've got certain visibility into cost, and it requires observability and so on. So that could facilitate the whole introduction of looking into what's going on in production for you.

[CHUCKLES]

STEVE: Great. This is, again, maybe a leading question, but I'm curious. One of the ideas that SRE teams have going into adopting-- or maybe it came from the books. It's hard for me to really know where this idea came from.

SRE teams, in some sort of platonic ideal state, will magically upgrade the guts of some system. The term that we've used in the past has been, like, to repair the tires on a race car during the drive or whatever-- or repair the wing during flight or something like that. How often does this actually happen?

Do you see teams specifically from an SRE either directed-- hey, we should do this, or directly from the SRE team itself, actually swapping out the guts of some running system in such a way that it continues to run and scales up, and is faster and better and shinier and happier? Is this a myth? Is this real? What do you guys think?

AMY: I think it's a bit myth and a bit real. I think SRE teams-- and I jumped in ahead of Vlad because I know he's going to have a ton to say about this. I think platform teams, like the SREs that have established a developer experience that they own, especially when they get the abstractions right, can change out everything underneath it.

And nobody ever knows. Like, if the abstractions are really solid, that's exactly what they're for. And so if you have Kubernetes in place, you can swap out the Kubernetes fleet a lot of times, even entire clusters, and nobody even knows.

Sometimes the software engineers don't even know you did it. So I think it is real, but it has a

scope. It pretty rarely, in my experience, spans past the infrastructure layer. And then once you get up into the application tier, all of a sudden now, we have so many product requirements involved-- critical user journeys and things like that, that you're back into a plain old software engineering effort to go in and change out components of the software and move it around. And that's where the SRE comes in more of an advisory role.

And I do this a lot these days where I come and help a team figure out how they're going do Strangler pattern, which is where they basically grow a new API, move all of the traffic over, and then kill the old one off when it basically has no more light. And usually what I do is, I go just help people reason about, like, here's your two-year plan to get off this very old crusty platform with a million edges that are going to take forever to file off. And here's how you can do load balancer tricks. Here's how you can do various-- whatever the tricks are we bring to the table, usually, we have those, especially for the infrastructure to stitch things so that we can do that seamless swap.

VLAD: And what I found is that this kind of hot swapping the tires on a Ferrari when it's racing on a track, that happens, I'd say, more often on less mature teams in terms of operations. On the teams where you've got more maturity that happens, I'll say, rarely.

AMY: Would you say it's rare or it's just constantly in motion, but smaller changes?

VLAD: Right. Yes, that's right.

AMY: I think that would be the maturity. It'd be like, an immature team is going to do like these big chunky changes that are dangerous. And when I see a really high-performance, mature team, every single sprint almost, they're pushing out some incremental change to move forward.

VLAD: Yeah.

STEVE: Great, so now open-ended questions. This is the fun part of the interview, hopefully. What would you like to see happen in the next two years when it comes to SRE around the world?

Like, people who identify as SREs, like official books, what is the big change you'd like to see? Or the even better, like, several small changes. It doesn't have to be big changes.

AMY: I would like to see our salary double 10 times.

[CHUCKLES]

That'd be good for me, anyway.

STEVE: Yeah. When someone offers you more money, you say, yes.

[LAUGHTER]

VLAD: You remember that part, you rename the team and give them a raise?

AMY: Yeah, I think if we asked all the SREs in the world if they would love this plan I just came up with, I think they would love it. But probably, we need something a little bit more practical to get the business to buy into it. I think this march toward platform engineering is going to be really good for SREs. I think as especially the enterprise crowd tends to follow where the herd has already gone, that's what I mean when I say enterprise.

It's like, they're not typically adopting the bleeding edge way to do things. But they do have this concept of platform already. A lot of times, it comes out of IT, and so we feel weird about it.

But I think there's a real opportunity here to build that basis of influence. Because you own the platform, we as SREs typically have the expertise to build those platforms and build them the right way. And that leads us to the influence in the leadership community to start to advocate earlier and earlier for reliability practices. So if I have anything to change, I'd just want to keep going down this road we're going, and just keep hammering on this to see where we can take it, to enable more SREs to get to the table before it's already in production.

VLAD: Yeah, so what I'd like to see in the SREs space is just plain application of AI for banal tasks. So if you're in a hurry today, then you need to have so many things in your head. Where are my runbooks? What is the SLO for the service?

What is the historical SLO adherence for that service in that region and so on? Why don't we do this just by having a proper chatbot that can give me all the answers like this. And also imagine how the onboarding of a new SRE would be simplified if the new SRE wouldn't have to bother the next table, but just would be able to have a conversation with some absolutely standard kind of chat bot that would know that stuff.

STEVE: Cool. And then speaking of new employees, what advice would you give to not brand-new employees, but even before that-- like high school, early university folks who are maybe looking in this direction? Like, hey, do you think anybody even knows this career exists outside of once you've gotten started in industry. But be like, how would people best be suited spending their time if they wanted to head in the direction of SRE and reliability?

AMY: I guess the main thing I would tell these folks is, there's one kind of work that's safe from AI. It's really just one that we have a fairly good notion is going to be safe for a very long time. And that is adaptive work, where we're in the territory of known unknowns and unknown unknowns. This is the place where AI just starts to fall apart, and where human creativity and pattern-finding and stuff starts to be superior. I think even to tomorrow's models, I think we will

continue to be superior at this.

So from that perspective, I think a lot of young folks are asking themselves today, like, what can I do that's safe from the machines? And they should be asking themselves that because we're seeing some real innovation in this space, I think. Where it goes, I'm not going to guess. But what will remain is that the computers will run into situations that they can't solve anymore. And that will be the wall where it's all the stuff that's already been done.

And so that leaves us. It leaves SREs. We're the ones that are in there fixing the stupid AI computer when it gets stuck. And say we buy an AI application from some vendor and it's handling a lot of our day-to-day stuff, like Vlad described. A lot of the low-hanging fruit, the stuff that's toil should be done by the computers.

What's left at the end are the unknown unknowns and the known unknowns, and some of the known unknowns. And where I would focus is being flexible, being adaptable, being somebody who's learned super-duper fast. Those are the skills you need to keep ahead, because the world underneath all of this stuff is changing so fast right now, I don't think it's going to look the same in 10 years.

VLAD: Yeah, I think so. And what I would suggest is, because the whole operational aspect is typically underrepresented in the university curriculum, it's important to get out of university and just get a job at a company that really runs something. And while you would learn something at university for the dev part of the story, you are likely not learning there anything substantial for the ops part of the story.

And you can learn the ops part of the story definitely in a company that's doing operations. So that's the advice that I would give. And then just follow your instincts. If you are interested there, you can pursue your interest there.

AMY: And that's how most SREs get here. Steve, how many of your colleagues are physics majors at university, half?

[LAUGHTER]

STEVE: I was going to say more like 15%, even 10%, something like that.

AMY: But it's a significant number, right?

STEVE: There's physics, philosophy--

[INTERPOSING VOICES]

AMY: Physics, math, philosophy. But that's the thing I think, though, that leads people here. It is

that curiosity. And then getting in and seeing the real world and be like, wow, this is how things actually work, that's where I want to work. Or I need to get paid.

STEVE: Yeah, my last boss majored in drama, I believe, or theatrical studies. So there you go.

AMY: I was a music major.

STEVE: There you go.

VLAD: Cool.

STEVE: OK, well, thank you both for coming today. This was a great discussion. Before we sign off, I think it's customary to mention where you can be found on the internet if people want to follow you. And maybe like one final take, if you're so inclined-- one parting shot. perhaps.

VLAD: You can find me on LinkedIn, and pay attention to observability.

STEVE: Nice.

AMY: I can be found on Mastodon @renice-- so like the command renice, @renice@hachyderm.io. And I can be found on LinkedIn as well. And my parting take would be, I think where all of our work is heading is what, in the resilience community, we call adaptive work. And it's that stuff that remains when everything's been automated, the stuff that the computers haven't predicted or the computers haven't predicted. And so really, site reliability engineer is a lot of ways are adaptive capacity engineers.

STEVE: Great Thank you both. And as we like to say, may the queries flow and the pagers stay silent.

AMY: All right.

VLAD: Thank you.

STEVE: So long.

PERSON: You've been listening to Prodcast Google's podcast on site reliability engineering. Visit us on the web at sre.google, where you can find papers, workshops, videos, and more about SRE. This season's host is Steve McGhee, with contributions from Jordan Greenberg and Florian Rathgeber.

The podcast is produced by Paul Guglielmino, Sunny Hsiao, and Salim Virji [INAUDIBLE] The podcast theme is "Telebot" by Javi Beltran. Special thanks to MP English and Jenn Petoff.

[JAVI BELTRAN, "TELEBOT"]