[JAVI BELTRAN, "TELEBOT"]

SPEAKER: Welcome to season 7 of the *Prodcast,* Google's podcast about site reliability, engineering and production software. This season, we are exploring what it takes to ship software and what that even means to different folks. Join us as we discuss how to ship it as SREs. And of course, the most important part is the friends we made along the way. Happy listening, and may all your incidents be novel.
[JAVI BELTRAN, "TELEBOT"]
STEVE MCGHEE: Hey, everyone and welcome back to the Prodcast. This is Google's podcast on SRE, Site Reliability Engineering, and production software. I'm still Steve McGhee. I continue to be. And I think we also have our good friend Matt.
MATTHEW SIEGLER: Yeah.
STEVE MCGHEE: Matt, how are you doing?
MATTHEW SIEGLER: I also continue to be. And Matt, yes, that's me, I'm Matt.
STEVE MCGHEE: I was going to say if you continue to be Steve McGhee also, we're going to be in trouble because that's going to be confusing.
MATTHEW SIEGLER: We continue to be different people. And that's great, because now we have two hosts. Awesome.
STEVE MCGHEE: All right. Much, much clearer. And we also have a third entity with us today. Third entity, why don't you introduce yourself? Who are you, and what's going on.
VASANTH KRISHNAMOORTHY: Hi, folks. My name is Vasanth Krishnamoorthy, but for low latency communications in the theme of this podcast, you can shorten it and just call me Vasa. I lead web foundations at Pinterest. We make the platform faster, more reliable, and also easier to build on, so product engineers do not have to worry about the foundational pieces and can build experiences which help our Pinterest users get more inspired and build the life they love. So my team broadly owns things such as frameworks, developer experience, design systems, and enabling adopting modern best practices safely.
MATTHEW SIEGLER: Cool.
VASANTH KRISHNAMOORTHY: And yeah, I'm really looking forward to this conversation.
MATTHEW SIEGLER: As am I.
STEVE MCGHEE: Me too. We apparently are experiencing this podcast in the future because all of our topics seem to revolve around AI. I don't know if you've noticed that, like every other podcast in the universe. And so let's start with that. Are you guys using AI at all? I'm guessing the answer is yes, but like, how? How has it affected you guys at Pinterest? Like, a lot, a little, in specific ways? What do you think?
VASANTH KRISHNAMOORTHY: Yeah, absolutely. I think as you said, pretty much the entire industry has been adopting it. I feel AI has made generating code much cheaper. However, the architectural judgment and also how we are able to reliably validate those changes and those things haven't quite changed. Also, the human accountability piece still remains. People still own the problems, the trade-offs made, and the decisions, so those things haven't changed. But definitely, AI has become a key part of how we are thinking about software development life cycle.

One of the things which we have noticed is definitely at least on the web side of things, we have noticed code generation has pretty much in a way doubled over the last few months. However, we're seeing the bottlenecks shift a little bit more down the funnel, especially when it comes to the code review phase and also the CI and validation. So generating code is cheaper, but validating those changes and rolling it out in a safer manner is the challenge, I feel.

And again, these are all areas where I do feel across the industry we are grappling with all of these challenges. How do we not just think about speed as generating code? And I really view speed as a system property from that lens. So it's really like generating code is faster. How do we make sure CI scales not just from an infrastructure perspective, from a safety-- are we able to reliably run all these tests? And also the code review piece, how do we review a lot of code coming in there safely? So those are all the things that we are working on.

Certainly, Pinterest has leaned in, in this case. However, we are similar to the industry thinking about how do we make the entire software development life cycle scale with AI, as well.

STEVE MCGHEE: So one thing that you mentioned earlier, which I want to take advantage of, I guess, is I think you said you manage not only the web foundations, but also the frameworks and the design skills, what did you call, the design--

VASANTH KRISHNAMOORTHY: Systems.

STEVE MCGHEE: Yeah, systems. And how is working with that massive generation of code? Like, are you confident that the code that's generating is using that design system? Like, is it actually consistent with what you would have done with 10 times the human engineers, or is it differing, or is it amending the design system in ways? How do you see that working?

VASANTH KRISHNAMOORTHY: Yeah, we are definitely experimenting with a lot of approaches here. On the framework side, there are multiple pieces, a couple of pieces. One is adopting frameworks. I think that part, one of the key challenges with adopting modern frameworks or better practices has been the migration. And I feel like that is a big opportunity for us to accelerate migrations, compress timelines with using AI.

We used to use code mods, automation in the past, just as an industry, and now viewing it from the lens of how can and where can AI actually help in these migrations or modernization efforts. I feel from a platform team's lens that is one of the bigger opportunities, more so than product generating different features using AI. Because in a way, if you think about it, most of the migrations, we try to either maintain or improve how the system is. So even if we were to get to where we are-- from where we are right now to where we want to go with the same features or whatever, it's still a big success in that lens if you're able to move to a better framework or so on. Those are areas where AI can help.

The validation part of it is really something which is still-- it really depends on the part of the code base or the system, how well tested, do we have the right code coverage in terms of testing and everything. And if those are there, then we can lean a little bit more there. If not, how can we generate those tests so that we can improve our validation? So the frameworks modernization part is one.

The design systems angle, it's really like building blocks with which you can build UI. Generating these building blocks, I feel that is definitely an opportunity. And using these building blocks, also, and building features, there is an opportunity. Again, early experiments around these, can we actually go from a design to an actual code. Those are all something we have experimented with internally on web, and we've seen it successfully. However, scaling this approach, figuring out where those bottlenecks are, where can we do it successfully, where we cannot, those are all problems that we need to continually improve.

So I see it less so as a binary of, like, have we already adopted this successfully. I think it's more like we are moving in the direction. We are thinking about agentic software development lifecycle more broadly on web, and also where are those bottlenecks and where are those edge cases which we should continually improve and make use of. So yeah, that's where we are in our journey right now.

MATTHEW SIEGLER: I like how you described that. First of all, you used the word journey, rather than an arrival or switch. And that makes me want to step back-- maybe not just step up

in scale, but also step back in time and ask you, what was this adoption uptake like for you, going back in time to 2024, 2025? Because there's a lot of skepticism as well as hesitancy to adopt this in a way that was going to work for you, to bring about more rapid change of code base, as well as production changes, but also do it safely and integrate it into your work streams and then give you these benefits you're talking about. So talk about the cultural change and, if not, tooling changes that you had to do this safely. And also bring to you where you are now.
VASANTH KRISHNAMOORTHY: Yeah, definitely it's a journey, but I think they're all converging in the sense that the models are getting better. We know better on how to manage context. So things are moving in the direction. And on the cultural aspect, I think it is good to encourage all of these different ideas, people who are skeptic about AI, people who are leaning in. Certainly for the early adopters, we want to encourage them to try out different use cases because as an industry, we still haven't aligned on what is the best way to use AI.
So trying different things is a good way for us to do this internally in the company. We have tried out hackathons and stuff to experiment with different ideas and also-- but investing both in paved paths as well as alternative routes for people to innovate in their own sense and figure out, could this become a best practice for us internally at Pinterest?
So we've tried out all these things. Culturally, I mean, speaking more specifically within my own team and web foundations, I really viewed any kind of cultural transformation as something where you want to encourage the early adopters to really try out things wild. I think I read this famous quote, I believe, by Simon Sinek, who mentioned the laws of diffusion of innovation, where he mentions like, you never get people to do a cultural transformation by asking everybody to try something or mandating something. You really focus on the smaller subset of people, initially, who are leaning in and really the folks who you feel are like the ones who could be the flag bearers for this. Encourage them to try it out and show how it can be successful. And that is the best way.
And then you have this bell curve where there is a lot of people who are in the middle of the bell curve, which is like 70% of them, or 50% of them who are roughly like, I am still in the middle. I don't know is it this way or that way? And then when they start seeing early adopters really showcase the potential of it, they're like, oh, yeah, I want to try it out, and I want to learn more about this. And there are actually the bottom percentage who might be a bit more skeptic, which can take a little bit longer time. But it's really to think about the top 30% who are experimenting with this and enabling them to try out more boldly.
Internally within Pinterest, this means more about thinking about problems that we have at a platform level. Or we already have projects, migrations in progress, all of these things, how do we accelerate this with AI? Is there a way to do this safely? And giving them the bandwidth and also the tools and leverage for them to be able to try this out. And if it's successful, it's great. If it's not, we still have a learning. And that is something which we have imbibed into our culture of win or learn, which is a value we have in Pinterest. And that's really helped us in imbibe this. And also, how leaders are using it can also be some way by which we can encourage engineers also to adopt it. So modeling the right example at different levels is also another key area.
So that isn't one single size fits all approach to do this, but I will largely view this from the angle of encouraging early adopters, encouraging dialogue for both ends of the spectrum on where things are successful, where things aren't, while also recognizing, tying it to how does this help our product more broadly get better. And tying it in a way to the business outcomes or the goals of the company can be a good way for us to see it. But I think we've really crossed this path where at this point, pretty much all of the engineers, non-engineers within Pinterest are leaning into AI in some form. And I think that's actually a great thing.
MATTHEW SIEGLER: So you're describing your organization as pretty experimental in nature,

and giving the freedom to come out to take different levels of individual personal risk with tools to see how they play out, then taking results that do work or don't work through experimentation, through studies. Now, tie that to the problem of reliability. How do you explore the problem of how do we make our work more reliable through experimentation? They sound like those are in opposite directions, but I imagine they're not. So what are we doing to make things for the user experience better, but sometimes that requires taking risks with the software or the product as a whole?

VASANTH KRISHNAMOORTHY: I feel this is an area where we want to set our standard for what good looks like when we push something out there to our user. And we already have those guardrails, or like even before AI existed, these are the things we need to test. This is how we make sure we are validating it's good enough for the user. Really thinking about from a user lens is one way by which we can think about the product's quality.

And ask the experiment, are we meeting the bar for quality, be it like having our CI checks pass, having our manual verifications valid where required, to bridge any gaps we might have rolling out in Canary, staged rollouts, progressive validations, all of those things which helps us validate if this quality enough still remains.

However, you can also experiment with how do we develop code. And there is things which are behind the scenes of what-- there is one audience, which is the users. The other audience is the developers. I think the developer audience is where there is a little bit more, can have some friction in the sense that there could be a code change, which could work fine from a user's perspective, but it could be like, hey, this does not look it's a scalable approach or it's the best practice for how we code. It looks very different from the rest of the code base. I think those are the areas where we will need to lay more guardrails around best practices.

Earlier, we relied a lot on a human in the loop to review those changes. But as changes became huge, you can quickly generate a lot of lines of code. Recently, I saw this term coined by Tobi from Shopify, who's the CEO of Shopify, called slop grenade. So huge changes which look-- plausible changes. They work, they look good. And then you might approve it. But they are also risky, and their blast radius can be bigger-- sometimes very hidden. Like, it could be through memory leaks and stuff.

So those are the areas where we need to really contain and see OK, can we ship smaller changes? Can we have somebody self-review the code, which in the past wasn't something like you wrote the code, and then you had someone else review it. But then can they also review their own code which is generated by AI? Which ties back to the earlier point around AI can generate code, but you cannot necessarily make human accountability cheaper.

People still have to own the problem, the trade-offs made, and everything. And I think that is the part which I feel is really important for us, to make sure this whole approach scales. So still, it doesn't mean like the point is to slow down AI. It is really to make it safe enough so that we can scale these workflows in a manner which is really safe.

STEVE MCGHEE: Speaking of accountability and reliability and things like that. So like I presume that you guys have mechanisms in place.

So one thing that you've mentioned a couple of times is this idea of migrations. And you have--

VASANTH KRISHNAMOORTHY: Got it. Yeah.

STEVE MCGHEE: Also, you must have lots of forms of telemetry and you have release management controls and things like this. So given all these controls that maybe you've-- I mean, Pinterest is not a brand new company. It's been around for a little while. So I'm presuming that you've been like adopting and adapting these over time. What do you think has been the number one or your favorite change that you think has been the most effective to those controls in this new era when we're generating all this code and we're still like aligning ourselves

to the frameworks and the platform and everything?
But did you find that there was something in particular in these more traditional, I guess, mechanisms that was really useful or that needed to be changed in some way to become really useful in this new world that allows us to give this responsibility to individuals and trust that it will work, that they will actually deliver code that is safe?
VASANTH KRISHNAMOORTHY: Yeah, definitely. I think those, I view them largely as systems to help us give confidence that our code is reliable and it's safe to release. And they have-- if you see from a software development lifecycle perspective, they have evolved, but not necessarily completely changed in how we view those. For instance, we have been trying to shift left more in terms of how we validate these changes.
Earlier, a few years down, I think across the industry, it used to be like, don't worry about it, things, just ship it and then-- I mean, essentially move it to the CI and then you will be able to get a sense of what's failing, et cetera. Of course, we had our local checks. But CI was the primary way for us to really have a thorough testing done in terms of whether this is actually good to go into the main branch. The key signal is also like, are we able to roll back quickly and revert if there is something happening.
The other part, which is also at least on the web we've been blessed with, is just as a platform, we can roll out things as quickly as we want to. So we have a continuous delivery system, which means we don't have to wait for a huge amount of changes to go in at one time to really figure out what's wrong. So it's across the whole cycle. And on top of it, even at a review stage now, we are able to leverage AI as a first pass for code reviews. Different models, if it was generated using one of the models, you can have an adversarial review through another model and really see, is it able to catch any issues. So that is one part of it.
The other part which is evolving is really at a code base level. Like earlier, some of the changes we've done on web is earlier we used to view documentation separately and then code separately. Now we are trying to co-locate those. So as agents are able to look into the changes to make, it is also having documentation co-located. It can understand better about these systems. And having defaults, like files like AGENTS.md, how should we process this repository, what are the dependency graphs. Anything we can feed it as context for AI to keep it in its lane and not diverge too much and go wild can be really helpful for the type of changes we want to make. Those are all things which we are continuing to improve on and figuring out what are those things.
Those practices are evolving as an industry, as well. Recently, there was a debate about Claude.md, AGENTS.md. So things are evolving in this space. So it's important to keep a watch for what's needed for Pinterest code base, what is happening in the industry, and evolve this accordingly, as well. But the key insight, that is earlier documentation was viewed largely for engineers. Now, I think the documentation and the context management piece for agents and humans is going to be equally important in this phase for us to go through all these steps. It's kind of building accountability and systems at different levels to give us the confidence. But ultimately, as I said, human is the one who's accountable. And there are also different teams which handle these different sections of it. And that is how the accountability is divided.
MATTHEW SIEGLER: Can you relate an anecdote about where AI-initiated or agentic work actually led to developer improved documentation or APIs, or removing obsolete code, or working upstream rather than downstream, like refinement and iteration?
VASANTH KRISHNAMOORTHY: Yeah, certainly. As you mentioned this question, pretty much all the things you mentioned, it has really helped out in this case. Again, I would say not necessarily autonomously. One example is we in the past writing documentation, all these engineers find it like a really hard thing to do. Now, we are able to guide and/or navigate AI to

can you really understand this part of the code base and generate a draft for how it works, even to some extent like increasing it to build a Mermaid diagram of the flow of how this part of our codebase works?
And you still have somebody who's an expert in that codebase really like review and say, this one is like completely wrong. This one is right. But it makes the whole process a little bit more less strenuous on, oh, I need to write this documentation from scratch rather than navigate it or steer it in a direction which can actually help. So this is one way by which we actually improve documentation. We've had internal doc-athons or ways by which we try to improve our documentation. This, of course, helps with onboarding later on as new people come in, as well. So that is one way.
The other one in terms of how we are thinking about quality, as well. So that is we have automated tests, but we also can generate automated tests with AI. Again, we need to provide the right guardrails. And this is something not necessarily my team, but some of our peer teams have focused a lot on, like how do we improve the overall quality, but find the right balance? You don't want to just add coverage for the sake of it with various tests. How do we improve test coverage in a thoughtful manner and have someone validate if it's high quality enough?
So it's able, given the pace at which we can actually generate these tests, the challenge is really how do we steer it in the direction it's actually building confidence versus building coverage alone. And that is the part which several people have worked on to improve overall coverage, because we do feel like if we have an application which is well tested and battle tested, making these broader changes of framework adoption or migrations becomes much easier. So it's largely bridging the gap where we might have certain areas, which is not as tested well enough.
And also in terms of improving how our application already works, one example I can point out is we have used it in triaging certain issues as we run into some incidents. It's able to do an initial analysis, even for somebody who's looking into the incident, even before they started to dig in. Of course, it's able to-- you can ask questions and it will try to answer internally on OK, this is failing. Is it because of this or is it because of that? What do you think it is? And then it learns through all our different incidents happening across the platform.
So really, it's thinking about-- it has to be done thoughtfully. But also I feel there is an opportunity for it to help up the funnel, but less so of like here is something which you completely missed in our architecture, which AI completely transformed. Perhaps it could differ from system to system. I'm again speaking a lot, largely, about web. I know a lot of ML teams have been using it more aggressively in terms of how it can make the product much better. But in web, these are pretty much like the areas across the software development life cycle, how can AI actually help. Also, moving some of these improvements up the funnel, not just in moving it down the funnel to ship?
STEVE MCGHEE: Yeah. One thing, so we have a little like document that we shared ahead of time with ideas to talk about and things like this. One thing that really caught my eye is you talked about you guys did a migration from Flow to TypeScript. I presume that's like-- that was a big deal. It sounds like it was a big deal at the time, if not, if it's still going on. But one thing you pointed out in it is that you had a decision around a big bang versus a progressive rollout.
And then-- Can you talk through this in a little bit of detail of how do you make this-- like, to me, I would say like always try to be progressive. But it sounds like you guys found that in some cases, actually a big bang made sense for some components, perhaps, or for some subsets. How do you make this decision? How is this-- are you looking at reliability first or are you looking at speed first, or things like this? Does any of this relate to AI, or is this just like the traditional SRE work? I'm curious. It sounds like a good story, so I'm hoping …
VASANTH KRISHNAMOORTHY: Yeah, absolutely. This happened a couple of years back. The

migration is complete, but it was one of the bigger migrations we did. And then we also published a blog post on this. I think the decision there-- and you're right, progressive migration or gradual migration is usually the best practice. But I think I view best practices more as starting points, not necessarily like commandments. It's a good opportunity for us to think like, is this best practice still the best practice for this particular context or migration?
In a way, I would say best practices in a way are like average practices. The common approach works in many situations, but we still need to choose what is best fit for this migration. And in this particular case, it was really a question of Flow and TypeScript or a couple of ways by which you can add typing to JavaScript. So one is like annotation, the other one is kind of a superset language.
So moving from Flow to TypeScript was the key project we had. And we were really considering, should we have this gradual path where we support both Flow and then Typescript? And that is not quite straightforward. We will have to add a lot of piping. And if you were to do that, it would essentially expand out our timeline for actual migration, because we will have to do this a little bit more gradually versus what if we actually did it as a cutover? What is the cost that-- obviously, there is a cost. Like, you might have to have a small period of time where you made it lock out the repo or essentially make it like-- is that cost worth it? I think that's the question.
And I view these migrations from two different angles, at least from a platform migration perspective. How does this impact users, which is our end users? And also, how does it impact developers? I think for this particular migration, these are types and typing to make the quality of code and the product better. So from a user's perspective, the baseline is we shouldn't see any degradations or regressions. If anything, we may want to identify issues and fix them and make it better. So we pick that as a baseline.
One of the smart things we did was like, if we were to strip out Flow and TypeScript, can we do a byte for byte comparison and see what is differing? Like technically, they should be very similar, if not the same, ideally. So that is something which we baked into one of our validation processes, to make sure the code which we are pushing out is reliable.
The second thing is we also thought about would it make sense for us to-- one other thing which is counterintuitive we did was we did this migration on a Friday, not necessarily from the angle of Friday deployments are scary, yes, but putting it from the user's lens, if something went catastrophic, traditionally our users, we have lesser traffic on the weekend for the product. So how do we-- and so any disruption could be contained in that sense.
And also, if you wanted to lock the repo we could do it on a Friday evening. Again, I'm depending on the audience. We have developers. Majority of them are in the US. And that makes it a little bit easier for us to figure out what is the right timing for us to do this. So those were the reasons. And we felt like we could do a lot of validations in our own branch around is this TypeScript version working the same way as Flow would?
We pretty much did 30 to 60 days of continuous getting the latest code from the main branch and then converting it to TypeScript, running all these automated checks, manual verifications, and essentially battle testing our approach. A lot of this was just automated. So we didn't have to do this every day by hand. It was more like running a job to get this done and validate it. And it was really boiled down to when is the right timing for us to do this cutover? But it was still a very massive migration. Around 3.7 million lines of code moved from Flow to TypeScript and doing all of this.
And there is also the other piece of it, which is less spoken about. The code migration is one part of it. The user impact is another one. But how do people be productive in this new language, Typescript? And that is where a lot of education-- and so that is also a key consideration. Like, do we want to optimize for people? Once we move to Typescript, are they

going to be productive enough.
We could have a very quick and fast cutover. But if people are going to be struggling with this new language, that could be a challenge. So we really timed it in a manner where we had a lot of sessions for training, helped them get up to speed, had a playground where people can write code in Flow and then try it out on TypeScript, as well. So really, a multi-step process.
One thing which really helped us is in general, as we approach platform problems, we tend to think about what is the right way to solve this, and that goes to a lot of discussions. And once we land on yes, this is how we are going to do this project, I think we get quickly moving. And we might adapt it a bit or evolve a bit, but largely we are focused there.
But the key thing I would say from a reliability standpoint is to really think about where can things fail. So really starting with the question of if this project was to be failing, what would cause that? And that is where we did a pre-mortem of sorts to figure out where would-- if this project was to fail, how would it be?
And that helped us uncover a lot of things. Maybe this dependency, maybe we need to really focus on these teams where we need to do a bit more of training than what we would do. How about new engineers who are coming in and joining? How do we validate this piece of it? So it really was thinking through all the unknown unknowns. There is still some learnings to be there, which is where we want to battle test it from the angle of like, is it easy to revert, all of those angles. But really, all of this combined really helped us in complete this migration with zero downtime. And for the user, for the developer, a little bit of like a pause where we were like, let's do this quick cutover instead of a gradual one.
And overall, engineers also liked it. One of those migrations where in our survey after the migration, we got 97% of the people, developers say that it was a very successful migration for them.
STEVE MCGHEE: Wow, pretty good. OK, this is awesome. You guys have a great product. I'm glad there's someone like you that is taking care to make sure that the platform itself is robust and usable and modern because I can imagine it wouldn't be as great if you didn't. That would be a real bummer for your myriad users.
So any final topics that you want to throw out there, like places we can find you on the internet, or final bits of wisdom that you want our SRE-adjacent listeners to be aware of, or favorite pinned item, perhaps, something like that? What do you think?
VASANTH KRISHNAMOORTHY: Certainly one thing as I think about reliability is I feel as I dug in more into the SRE aspects of it, one of the things which I learned a lot is really how we think about failure modes, how do we think about instrumenting it the right way. And that is certainly something which we have tried to imbibe on the frontend as we think about web client reliability, as well.
Similarly, on the other end, if you were to zoom out and say reliability is yes, there are metrics, there are guardrails, all of these things. But ultimately, we are trying to answer the question of is the user who's on the site actually who's coming here to do some task, are they able to do it successfully?
And that you could have an amazing client experience, but something latency on the backend side, which could cause them to be slow. Or it could be like we could have all the perfect backend metrics and available latencies or within our threshold spoil, the experience could be slow on the frontend side. So it's really viewing it from the client lens. I like to say a slow side is pretty much like an outage in slow motion.
It's not an outage. You hardly get pinged for oh, the site is super-- I mean, if it's super slow, probably. But what I mean is like performance is another key aspect of it, which is something we might think about all these backend services and the API layers. And at different points, are we

having the right latency and is it fast enough.
But when it gets to the client, every client behaves differently. They have different network connections. Things can fail differently on the client side. So it's really viewing it from that lens. We want to think about it as the availability piece on the client side, which is like, are we available? And of course, the user uptime, are users really able to use the site without any blocking issues as they go through their flow? For Pinterest, it's really a journey of trying to discover something, getting inspired, and realizing what they want to do.
And lastly, from a performance angle, is the pinner wait time good? Are the core web vitals scores looking good? So largely like the one takeaway I want to share is really viewing it through the full funnel from a user's perspective, both on the client site reliability aspect on the server side, and also how we manage any incidents which come from both of these aspects. That is the key takeaway, which I would like to share.
STEVE MCGHEE: Yeah, a holistic view of reliability is always good. It's hard to step back and see that, but I think it is always a good piece of advice. All right. Well, thanks for your time today. It's been great to hear a little bit of the detail behind one of the biggest sites on the internet, which is pretty awesome, and helps people every day. Matt, any closing thoughts?
MATTHEW SIEGLER: No, that was great. Thank you, Vasa.
STEVE MCGHEE: What's your favorite pin, Matt? What's your favorite thing on Pinterest?
STEVE MCGHEE: Bikes. Anything related to bikes, that's my answer. Whatever you got that has to do with bicycles, weird, fast, or slow.
MATTHEW SIEGLER: I like mountains.
STEVE MCGHEE: Mountains.
MATTHEW SIEGLER: Yep.
STEVE MCGHEE: Yeah, I'm sure there's pin boards for that. Cool. All right.
VASANTH KRISHNAMOORTHY: If you want to find me on the internet, you can follow me on Twitter. You can connect. My handle is Vasanth_K. You can leave it in the show notes or something. And also, you can find me on LinkedIn, Vasanth Krishnamoorthy, probably better spelled down than me spelling it out here.
STEVE MCGHEE: Oh, yeah, we'll put it in the notes. No problem about that.
VASANTH KRISHNAMOORTHY: All right. Yeah. Thank you for having me. I really enjoyed the conversation.
MATTHEW SIEGLER: Take care, Vasa.
STEVE MCGHEE: All right. Thanks very much.
SPEAKER: You've been listening to the *Prodcast,* Google's podcast on site reliability engineering and production software. 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 hosts Jordan Greenberg, Steve McGhee, and Matt Siegler. The Prodcast is produced by Paul Guglielmino and Salim Virji with contributions from many SREs behind the scenes. The Prodcast theme is "Telebot" by Javi Beltran and Jordan Greenberg.
[JAVI BELTRAN, "TELEBOT"]