EPISODE 840

[INTRODUCTION]

[00:00:00] JM: Netflix started with a DVD by mail product. The software infrastructure and operations practices needed for Netflix's DVD business were very different from those needed by the streaming video company that it is today. Since the early days of Netflix, CEO Reed Hastings knew that the company would evolve to become a streaming video platform, but he did not know when the technology would be advanced enough to support video streaming, and he did not know how users would consume it.

Greg Burrell has worked at Netflix for 14 years. Greg was one of the first engineers to start working on video streaming, which Netflix first attempted to implement with a setup set-top box that downloaded movies and played them on your television. After evolving this strategy, Netflix arrived at the current model of video streaming through apps on browsers and mobile devices.

As the company pivoted from DVD by mail to video streaming, Netflix encountered multiple challenges across engineering, operations and communications across the company. At the time, there were no dev ops practices. There was not established continuous delivery tools. The available cloud technologies were immature and low-level.

Greg Burrell joins the show to describe the evolutionary arc of Netflix's engineering process. Greg also presents a model for software development that he describes as full cycle development. At Netflix, engineering teams of full cycle developers work without dedicated operations or testing teams. It's a sophisticated approach to engineering management and whether you're an engineering manager or you're just somebody who is curious about human practices of software engineering. This is quite an interesting and useful applied software engineering show.

I spoke to Greg at the Fullstack Tech Radar Day, a software conference in Tel Aviv put on by Tikal, an engineering community based out of Israel and San Francisco. This was a great conference and we will be airing some additional content from that conference in the coming weeks.

A new Software Daily update is on iOS, and if you are a user of the app, you've probably seen this update already. It's a lot of improvements that we've made to reliability, downloads, bookmarks, community, search will be hopefully updated soon, and it's really just been a long term effort and we've invested a lot of resources into it. So you can check out the Software Daily app for iOS by finding it in the app store or finding the link in the show notes. You can use it to become an ad free listener if you don't like the ads of the show. You can support the show with $10 a month or $100 a year for ad free support, and there will be some additional benefits hopefully in the near future, and we'll be working on Android very soon. I'd love to get your feedback on the software daily app, the softwaredaily.com platform, to understand better what you would like to see out of the Software Engineering Daily community and additional features.

We're booking sponsorships. If you're interested in sponsoring Software Engineering Daily with advertisements, you can go to softwareengineeringdaily.com/sponsor. I guess it's weird to read that right after I mention that you can become an ad free listener with $10 a month or $100 a year, but most people are fine with ads, or at least they don't speak with their wallet.

We're hiring two interns for software engineering and business development. If you're interested in either position, send me an email with your resume, jeff@softwareengineeringdaily.com. Please put internship in the subject line, and don't be afraid to reach out. I'd love to hear from you if you're interested in working with us as an intern.

With that, let's get on to today's show.

[SPONSOR MESSAGE]

[00:04:22] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves

the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW]

[00:06:41] JM: Greg Burrell, welcome to Software Engineering Daily.

[00:06:44] GB: Oh, thank you. I am thrilled to be here. I've been a fan of this podcast and I'm just amazed, the fact that you can do this on a daily basis and always have such interesting guests on.

[00:06:54] JM: Well, thank you very much. You are definitely one of those interesting guests. You've been at Netflix for 14 years?

[00:07:00] GB: Forever.

[00:07:01] JM: Okay. So 14 years effectively forever. I want to start with the subject of Netflix's move from the DVD business or the expansion from the DVD business into the streaming business, which started with a small experimental team that you were a part of.

[00:07:24] GB: That's correct.

[00:07:24] JM: What was the initial scope of that project to evaluate whether or not streaming was viable?

[00:07:36] GB: Okay. Yeah, that's an interesting time in Netflix history. It was never a question that the future of Netflix would be movies over the internet. Reed Hastings has got on record to say he named the company Netflix for a reason. He knew there's going to be an end to the DVD era, and for a long time that was how he built the company and it got very successful on DVDs, but he saw the future coming. He knew it was going to be something to do with movies over the internet. So it was always a matter of timing when.

So I was brought on board in 2005. I saw they were looking for a tester. I didn't really know much about Netflix at the moment, at that time. I've been a subscriber off and on. I thought it was a good product. I liked it. But they were looking for a tester, and so I went and applied, and it was really interesting because they said, "We're not looking for someone for the DVD side of things. We're looking for someone to help out with the streaming side of things," and it wasn't even streaming to be honest.

At that time, it was still in 2005 and people most had broadband DSL. A lot of people had dialup. So streaming over that sort of internet just wasn't a viable option. So the initial model was really more of a download model, which ironically we just back to in recent years. But, initially, we were thinking download.

So when I came on board, we were building a hardware device, a hardware client device, kind of like a DVR, or TVO, had a spinning hard disk in it, 40 megs. You can download some movies. So the model was you go to the Netflix site, you find some movies you want. You download them overnight. Next day, you go look on your queue on the screen. You got three movies there. You can watch them. When you're done, you delete on and the next one downloads.

But we played around with this technology for a couple of years. We were building the client-side hardware, the client-side software, server-side software. At some point we realized streaming is possible. It's viable. The internet has caught up. Not everyone in the U.S. will have it back in 2007, but we knew it was coming. It was around the corner. So we ditched all the old download hardware, hard disk, and we went to the streaming model, where we would stream movies off of our servers straight into people's homes, into their devices.

[00:10:14] JM: So the efforts to build what was essentially a set-top box, were you just kind of doing those in parallel with watching the potential model of somebody being able to stream over the internet? I'm just trying to figure out, did the construction of that technology actually used for anything, or was it just kind of a prototyping experiment side of the company?

[00:10:38] GB: It's funny. When I was hired, they said, "This is the way we're going to go with our first initial offering of movies over the internet, and we're going to order a few thousand of these devices, boxes to be built, and we'll start from there." Well, we never got those few thousands boxes built. We had delays in the hardware, delays in the software, delays in the server-side of things, delays in licensing as well.

So at some point we also were running up into the issue of can we get this box out at a good price point where people will feel comfortable by one of these things? At the same time, we had a very successful DVD business. So weren't under immediate pressure to start producing profit from movies over the internet.

So we had a little runway and we want to get it right. We want to put out something that our customers, our members would be happy with from day one that would not get a poor reputation that people would get the satisfaction that they expect from the Netflix brand. Because, I'll tell you, one of the great things about working at Netflix is you go anywhere in the world and you mention Netflix and people's faces just light up. If I wear a Netflix shirt on public, people would be like, "Hey, Netlfix! I love your shows. I love –" and they'll tell me the shows they love. That's great. So we had a huge amount of brand love and we didn't want to do anything to damage that. So we played around with different technology for a while. We played around with this

client hardware bots and then we just got to the point where, “Hey! Streaming is viable. Well, let’s go with that model now.”

[00:12:22] JM: Okay. Once that happened, the entire company gradually began its shift towards orienting itself around that, because I think there was – Obviously, this is a period where plenty of the business was still around the DVD by mail infrastructure. So, gradually, you had to go through this process of migrating company resources from focusing on the DVD by mail business to the streaming business.

[00:12:54] GB: That’s correct.

[00:12:54] JM: And I assume you did that in parallel with the increased usage of the streaming platform. But it just must have been hard to juggle those two lines of business for the company as a whole for some period of time, right?

[00:13:09] GB: It was. It was a decision made that I think in retrospect was brilliant and that we gave away the streaming as an add-on to our DVD business.

[00:13:19] JM: Oh, yeah.

[00:13:20] GB: So you were a member and you got three DVDs at a time, right? We give you so many hours of streaming to go with that and we didn’t charge you any extra. It was an add-on. So the shift from disks to streaming was more organic as people discovered it on their own. They could still be plenty happy with a disk service and they love that, and, “Hey, at the same time, let’s try out the new streaming thing. Okay. Maybe offerings of movies are a little more limited, but we’ll find some gems in there, and it’s not costing me any extra.”

So I think this was a brilliant move to sort of just let people discover on their own and fall in love with it that way. Myself included, I realized at one point I haven’t sent back a disk in months. Without intending to, I’ve switched entirely to streaming myself at home for my own entertainment needs. So I think a lot of people fall to that same trajectory. They just slowly, naturally enjoy the convenience of streaming.

[00:14:20] JM: What a nice model for having a forgiving customer base early on, as if you just give this thing away for free, then you can very forgiving customer base, because they're not going to – I mean, they might still send in complaints like, "Hey, this thing is slow, or whatever," but you can always say, "Hey, we're just giving it to you for free."

So at a certain point, the company did really start putting a lot of wood behind the arrow of streaming and the engineering team started being oriented around this. How did Netflix engineering change as the company shifted its focus from DVD to streaming?

[00:15:02] GB: So, initially, as I mentioned, we started this movies of internet I'll call it as sort of an experimental group within Netflix, sort of a startup within Netflix. So we were a fairly small team of engineers. I myself was hired as a tester in this team, but of course as in any startup, everyone wear multiple hats. So I would spend a lot of my time setting up build automation, doing build and release engineering, doing standing up servers in our data centers, doing network experiments, all sorts of things like that.

So we worked in the sort of this startup mode while we were experimenting with the technology, while we're exploring all these different possibilities. At some point in 2007, early 2007, we realized, "All right, it's time to start shifting this more into a product offering that we're going to put in front of our customers."

So when we launched the streaming service. Again, it was a small scale launch. It was only available on the PC and you needed a plug-in for your web browser. The initial launch was I guess you could call it a soft launch.

[00:16:12] JM: Silverlight.

[00:16:13] GB: Yeah, Silverlight. Yes. You had to install Silverlight plug-in to your browser, and that was the way we got this off the ground. Along with that, we decided to shift how we organize our engineering team. Kind of get out of this scrappy startup mode and integrate with how the rest of Netflix did things. We were going to be part of the company. We don't want to be this little division off to the side with that. We're part of the future of Netflix. Let's integrate with the rest of Netflix. Let's learn, let's borrow from all those other great work that they've already

done on the DVD engineering side. Let's put that to use. Let's integrate with it. Let's teach them about streaming and let's learn from the DVD business.

[00:17:00] JM: So in the days of the DVD by mail business, you did have software that you needed to write, but it was mostly backend applications for managing the DVD by mail business. You didn't have customer-facing software. So I guess take me through the transitions in engineering practices that had to take place as the company was shifting from the DVD by mail business to this streaming business. What had to change in order to improve the engineering practices?

[00:17:33] GB: Yes. Our DVD by mail business did have a lot of customer-facing software. We had the website. We had the recommendations engine. We had a user experience we want the people to start off with in finding the movies that they wanted to get mailed to them.

One of the things about the DVD business was you pick three movies and you're sort of committed to those for the next week at least, right? They're going to get mailed to you. Maybe a day or two, you watch it. You mail them back, and then you may be pick three more. So we wanted to help our members find through really good movies that they'd probably really enjoy. So we want to make that experience easy. So we had customer-facing software in the website. We had the recommendations engine. We had the rating system. We had user reviews, all these things to help our members find something that they wanted to watch. So my point is we definitely had customer-facing software. They had really good solid engineering practices. They had development, testing, operations release.

What the change was the up and coming streaming side of things, we were in the startup, as I mentioned. Now we wanted to integrate with the rest of Netflix and get a little more rigorous. So we had development teams and we had teams that were now focused on testing. We had operations teams. Netflix, the DVD business, already had a central network operation center and now we had to have them start handling the streaming side of things as well.

So this was a good point to integrate with the rest of Netflix. So it was an evolution. We felt the need to really sort of align ourselves with the rest, with the DVD business, which, again, was still the main business of Netflix at that time.

[00:19:27] JM: What kinds of processes did the company have to put in place, or what kinds of software did the company have to build to support the changes of the streaming business? I know there were changes to how the teams were managed. There were changes to what kinds of infrastructure software was being built. Give me a perspective on how the engineering organization started to change after that shift to streaming.

[00:19:57] GB: Okay. That's a good question. Prior to that change, again, startup mode. Everyone does everything. I as a tester would be deploying our new build straight into production. I probably had set up the production servers myself in the data center. Got all the software and networking configured.

Once we shifted into more, "This is part of the business. This is the future of the business," we tightened things up a little. The network operation center was now in charge of the production systems. If I wanted to change, I put on a ticket. If we wanted to deploy new software, I'd open a ticket and the NOC would take the software and deploy it on to the servers. Our developers, they now just focus solely on development, less on maybe on experimental stuff. More on improving the product, getting out in front of our customers, staffing out our test teams so we can be sure we're putting a good, reliable product in front of our customers, and then the operations team, which is our network operations center, they handled all the production stuff. I no longer touch that in theory.

Some of the difficulties of the model was that if there was a production incident, our network operations center, they were these amazing guys that were staffed 24/7. They had these huge monitors with lots of graphs. An alarm would go off. An alert would happen, and they'd spring into action. Well, they weren't so as familiar with the streaming side. So spring into action often meant, "Let's call Greg. He knows how this stuff used to work."

So I would get that phone call and maybe I'd loop in a developer or maybe we'd loop in some database engineer. So there was a lot of friction in the system. I might not have, or the developers might not have full access to our production system. So we'd all be on this conference call at 2AM and saying, "What do you see? What does it look like? Can you try restarting the server?" Here's how you restart the server. Now, what do you see in the error

logs? Here's where you find the error logs. Do you see anything that looks error-ish?" The network operation center engineer will try and help out these things. Find something that looks error-ish and mail me 100,000 lines of log messages, and I might port through this. If I can't make sense of this, I'll forward that email on to the developer and she might look at it and say, "Well, try this or that." She could theorize about possible solutions. But without access to those production systems, it was really hard to validate those theories or to even test up potential fixes.

So troubleshooting productions and it's really became like this email chain back and forth. Our time resolution was high. It's not an ideal way to try and debug things via conference call at 2AM, right? So this was some of just the friction inherent in this model of trying to put ourselves in these teams and silo ourselves in effect.

So we operated like this for a few years, but after a while we realized, "This is not ideal. Something needs to change. We don't want to be in a place where – The usage of the streaming service is growing and growing. Customers are falling in love with that.

At Netflix we like to say, "All right, we're not a life or death service, but just watch what happens when the streaming service goes down for even a minute." You watch Twitter people's [inaudible 00:23:31] collapse and people would come in and panic. We want to keep our members happy. We want to be known for bringing joy and entertainment to people, not be known for outages and flaky service and unreliable streams.

So we realized, with this model of these compartments we're working and these silos, we're just running into too much friction. Our developers, they know the code, they know the applications. They're experts at that, but they're really not so familiar with the production system.

[00:24:07] JM: So these silos more specifically, were these – You had a silo for development and a silo for operations?

[00:24:13] GB: Yeah. We didn't call them silos, but yeah. Effectively, we had teams that were focused on each, and these teams were highly specialized and very efficient within each team.

But you're still trying to communicate information across these teams. Each team lacks full context, right?

So we tried things like handoff meetings. We do a handoff meeting from development, to test. Another handoff meeting from test to operations people. We would do weekly status meetings where we try to get everyone in the same room and up to speed on the latest changes. We would try change logs, which really just means you dump all your commit messages into one big text file and send that around and nobody looks at it.

We would try a lot of email communications really to try and get everyone on the same page and up-to-date, but things were changing so fast. We're growing. Our architecture was getting more complex. There are more services coming online. The effort to keep everyone up-to-date was just really a huge, huge communications overhead, introduced even more friction into this process.

[SPONSOR MESSAGE]

[00:25:26] JM: DigitalOcean is a simple, developer-friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy with an intuitive API, multiple storage options, integrated firewalls, load balancers and more.

With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow. DigitalOcean is simple. If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support, plus they've got 2,000+ tutorials to help you stay up-to-date with the latest open source software and languages and frameworks.

You can get started on DigitalOcean for free at do.co/sedaily. One thing that makes DigitalOcean special is they're really interested in long-term developer productivity. I remember one particular example of this when I found a tutorial on DigitalOcean about how to get started on a different cloud provider, and I thought that really stood for a sense of confidence and attention to just getting developers off the ground faster. They've continue to do that with DigitalOcean today. All their services are easy to use and have simple interfaces. Try it out at

do.co/sedaily. That's the letter D-O.C-O/se daily, and you will get started for free with some free credits.

Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:27:27] JM: I think this time that you're describing was experienced similarly by a lot of different prominent tech companies in the Bay Area, and I'm sure more abroad, and I think this was kind of the pre-dev ops days. This was sort of when people were starting to realize there's something wrong with the way that we're organizing our teams. It has something to do with the infrastructure tooling. This was pre-cloud, although also in the early days of cloud. Thinking back, it almost seems like there wasn't really an alternative to going through this painful period, because if you think about what we needed to get beyond these problems, the laundry list is quite long. You needed continuous delivery tools. You needed cloud arguably, infrastructure as code. You needed all of these different things in addition to a set of changes of cultural practices.

Was there anything you could have – I mean, I guess you did find a reorientation for the teams that was a little bit better. Also, as you were reorienting, that reorientation was taking place at the same time that the cloud was maturing. Is there any way to think back and was there a team structure that you could have done that would have actually solved your problems, or was this more a question of letting the team structures and the tooling advance in tandem?

[00:28:55] GB: Well, hindsight is always 20-20 if we knew then what we know now. Of course, we could have done something different. The interesting thing is we realized back at that time, this is not perfect. There's a problem here. We're running into a lot of friction. We've got teams that lack the context. We got a lot of communication overhead. We've got these really high MTTRs, mean time to resolution. We're losing feedback, right? Our developers are not so familiar with what's going on on production. Our production people are not so familiar with all the latest changes in our apps and our architecture. The feedback cycle is very piecemail and [inaudible 00:29:39].

We thought, “We got to try something different. This isn’t working. What can we do differently?” So we evolved to a different model, which we’re calling the hybrid model. In retrospect, at the time, it was called something new. In retrospect, it was more of a hybrid model. This coincided with our move to the cloud back in 2011. We had decided to move out of our data centers. We’re migrating into the cloud. That was mostly complete. We changed our model at that time.

Under this new model, we no longer have a test team. Our developers now took on testing responsibilities. They wrote code. They wrote tests. They ran tests. They knew how to test their code well, because they own the code. We spun up a dev ops team alongside them. Our network operations center, they’re focused on – They’re still on the DVD business and they know something about streaming business, but we need real operations specialists to directly address the needs of the streaming service. So we spun up a dev ops team. We also started to allow our developers to have access to production systems, because it’s in the cloud. If something goes wrong with an instance, they could just terminate it and get it replaced with a clean one. So our developers went on call after hours. They carried a pager for 2AM issues.

In retrospect, I’m not sure why we thought this would work. We had a dev ops team who was on call during the day and developers who are on call during the night. The problem was if a developer got called in the middle of the night, he or she might say, “Well, did the dev ops team make sort of change during the day that we don’t know about?” or the developer might say, “Well, I’ll just patch this over and wait for tomorrow for the dev ops team to come in and fix this correctly.

So, really, the split hybrid mode, it was an attempt to address the pain. But, really, they were still there. We still had this loss of context, this loss of feedback and we still had lengthy troubleshooting fixing cycles. So, again, we took the time. We stopped and said, “This isn’t working. How can we do this better?”

[00:32:00] JM: Was the entire streaming service in the cloud from day one?

[00:32:05] GB: From day one, no. We started out in our own data center. We had our own servers in the Netflix data centers. We had two data centers. We had built a second one for

failover purposes, and we were actually serving our content. We hadn't built our own CDN. We had five data centers around the country from which we were sending out the content.

So I and the streaming team built out those data centers and that was how we delivered content. But along with the shift of Netflix into the cloud, the streaming was sort of a vanguard of that as well. We shift boost. We did what we call in retrospect Roman writing, some of it in the data center, some of it in the cloud. One by one we moved microservices over to the cloud hopefully in a transparent way such that one day that microservice is coming out of the data center. The next day, we've dialed up the traffic to that same microservice in the cloud. It was a gradual shift, a gradual transition. We never made out, "Let's forklift everything all at once." So in this way, we could keep the lights on, keep that streaming service flowing. Keep our customers happy with the service, and behind the scenes, do this little migration in the cloud.

[00:33:23] JM: So the newer team structures that you started to implement, it sounds like those had to do – You're saying things like you would spin up operations teams to support specific engineering teams. That's what it was?

[00:33:38] GB: Yeah, we spun up a dev ops team. At the time, dev ops was a new thing, "We got to get you some dev ops." So we spun up a dev ops team. We realized there's a need for operational people who can really focus on the needs of streaming that's growing, that's changing, that's the future. So we want that operation team to really specialize in that. They will work alongside the overall Netflix operations team.

So this was an attempt to sort of ease some of the friction in the system to maybe if our previous model had a lot of friction between operations and development, well, we'll insert a dev ops team to try and smooth some of that.

[00:34:23] JM: Okay. So you spun up the streaming team. So after the streaming team started to gain traction, the streaming product was gaining traction. You initially had – Was it a centralized operations team?

[00:34:36] GB: That's correct. Netflix always had a centralized operations team for the DVD business and for the streaming business.

[00:34:41] JM: And for the streaming business. That led to problems, because once the streaming system started to have issues, started to have operational issues, it would go down. The centralized operations team would be trying to troubleshoot these issues and would not be able to troubleshoot them successfully. They would call up Greg, because Greg had started on the streaming team in the early days, and you would be kind of the source of knowledge. But that's just a choke point for solving all your problems. Eventually, Netflix said, "You know what? Let's just spin up an operations team that's dedicated to streaming," and that solved some of your problems, but it didn't solve all of your problems.

[00:35:22] GB: That's correct. I think one of the problems is this – As I mentioned, we tried to split responsibilities between developers and this dev ops team. Dev ops team during the day, you put out fires. You deploy a new software. You work on automation. Developers at night, you're on call. Well, each team is in some sense in the dark about what the other team did. Maybe the dev ops team would try and leave, "Hey, this is what we did today. Here is our status. So anybody on call at night knows." But the problem is someone comes out and a developer might get the page at 2AM. They don't know what the state of production was left in during the day. Who touched things? Who updated things? What got changed or reconfigured? So now they're afraid to act. They might make things worse. Better call some of those dev ops people. Even though they're not on call during the night, we better call them.

In fact, I was on the dev ops team, and being a natural night owl, I was usually awake. I tended to work through the night. That was just the way I worked best. I've just always been that way. So, "Hey, let's call the dev ops team. Bring them into the conference call and we'll all try and troubleshoot this."

In essence, we're back to where we started. We're back to getting everyone on a conference call to try and troubleshoot an issue and with that same high-resolution time. So we were attempting – To our credit and to the credit of some really good people at Netflix, we were trying something new. We were trying to change our organization model mid-flight. We were trying to address the pain points. It's just that this didn't really do that.

[00:37:08] JM: I think the next notable evolutionary point to address in the Netflix timeline is the development of more narrow platform teams. So, eventually, Netflix got to a point where there is an edge engineering team. There's a centralized, I guess, deployment team. I don't know what the exact teams are, but you basically got to a point where you said, "Okay, we're beyond just needing this reactive dev ops operations large support teams. We need to move forward to a more proactive stance. We need to build tools teams that can refine our infrastructure so that we can avoid some of these problems altogether, and in addition, build tools that allow us to troubleshoot and identify problems and surface information more proactively." Tell me about the transition to Netflix having these dedicated platform support teams.

[00:38:21] GB: Okay. Netflix always had some version of these platform teams. They provided libraries that were used by all of Netflix, both DVD business as well as streaming business. That was one of the changes we made when we took the streaming out of this experimental mode into a more integrated mode. We adapted a lot of the libraries and tools that the DVD business had been enjoying for years. So that was a good way for us, for example, to hook into the metrics system that the rest of the company had been using. That was a way for us to hook into a lot of the telemetry and base security libraries and things like that.

So these platform teams had always been there. What really happened was in about 2014, we realized, "Okay, this current model, the one I called a hybrid model, this is still not working. We still got a lot of friction in the system. We still got pain points. We're still trying to split responsibilities among teams. We have essentially different silos now, but we still have silos. Every change, every problem has to be communicated across those silos. Problems get bounced among different silo. Information [inaudible 00:39:35] inside those silos. How about we break down those silos?"

So we went to something that we now call our full cycle developer model. The change here is that our development teams, they now own all aspects of the software lifecycle. Our developers are responsible for design, for coding, for writing tests or running those tests, for deploying their builds into production, for operating the services in production, responding to alerts pages, doing performance optimizations, supporting partners. Our partners or others teams that rely on our services, and then background to design again.

So our developers now under this full cycle model, that they own the entire application. From design, through deployment, through operations, back through design and repeat with improvements. Repeat with bug fixes. So this puts our developers in control. They no longer lack context. They know the applications and they know our production systems, because they own both of them. Really, our feedback cycles are now as small as can be. Our developers are filling the direct problems, the pain points of operational issues. Now they're motivated, they're eager to address those pain points by redesigning applications, by changing code, by improving tests, by improving deployment, by creating new support tools. So that their priorities are really aligned around how can I run my service? How can I develop and run my service effectively?

[00:41:20] JM: Okay. So the full cycle developer that you're describing, is that only possible because you have these dedicated tools teams? Because you need dedicated tools teams to develop extremely good tools in order to empower the application developers, the people who are building random business logic to support the company. Those people who are building business logic to support the company, if they're going to need to also support the operations of their service, that means that they're going to need really good tooling to troubleshoot those service and to prevent those services from going down in the first place.

[00:41:56] GB: Yes, that's correct. When I describe this model to people, a lot of people react with skepticism. They say, "Greg, come on! Your developers, they're not installing security patches and tuning device drivers. They're not doing network reconfiguration. Nobody can possibly be an expert in everything." That's true.

We have central teams who provide tooling and expertise. We have a central team and we have automation that provides our developers with a latest, fully-patched Linux for them to then layer their application on. We have central teams for performance, for security. We have experts on those teams. They provide tools. They provide expertise. Our developers can consult with them.

So you might say, "Well, what if your developer doesn't know all the latest security best practices?" Well, we've got experts for that. We've got specialists who consult with us, who provide tools, who provide libraries with those best practices already built in. So our developers can really focus on the what, not the how. The what is building and operating highly available and scalable service that meets our members' needs, meet our partners' needs, that is

sustainable. This is not something we try and accomplish through a lot of heroics. We don't expect our full cycle developers to be super heroes who can do everything and they're willing to work themselves 24/7 to do it. No. This needs to be sustainable at our scale and our complexity. That comes through good tools, good tooling. Good tooling comes through the central tool teams.

We have strong partnerships with the central tool teams. We work with them. They learn about our use cases, our pain points. They deliver tools that need those, we feedback new pain points to them. When I say we feedback new pain points, I don't mean we open a Jira or a bug or shoot them an email. I mean, we sit down with the tool teams and we say, "Love what you've done with this latest version. Here's what could be better for us. Here's a bug we ran into. Here's a new use case that rose this past week." The tools teams, they want to hear that. They want to know that they're building the right tools. So they'll take that feedback and deliver something of even greater value.

So it's a mutually beneficial cycle, a bee-shaped beneficial partnership. So we have this really good partnerships with these amazing central tool teams, and that is the secret source that power – It's the tools, the good tools and the expertise. That is a pillar of the full cycle development model.

[00:44:49] JM: One difference between software development at Netflix today, and this is my perception. Correct me if I'm wrong. At Netflix today versus when the streaming business was getting off the ground, is you don't have as many outages anymore, right? The thing is like outages dealing with a service going offline in the middle of the night is so stressful and so disruptive to software development. In many cases, we've moved beyond those days. It's come combination of cloud and Kubernetes and continuous delivery and other modern tools that have really made the outage a thing of the past in many cases. From my point of view, that has helped software engineering so much, because it lets you take a less emotional perspective to a lot of these problems. Do you agree with that?

[00:45:45] GB: I think that's a great observation. It used to be that a developer –

[00:45:49] JM: I know you dealt with these outages. Actually, you talked about how many of these outages you've handled.

[00:45:54] GB: I've handled way too many of these outages in the middle of the night, sitting on my couch at home, in my pajamas with my cat sleeping next to me. I've been there. You're right, there's less emotional attachment. It used to be a developer would feel some – I don't know, anxiety about putting out new changes, about making major re-architectures, because what if the site goes down? What if I cause an outage? What if –

[00:46:20] JM: And it's Friday and I've got a date tonight.

[00:46:25] GB: Yeah, and I don't want to be the person that cause that frustration to millions of Netflix members. I want Netflix members to keep watching their favorite shows. So it is as you say, a combination of all these practices that continues delivery, gives us much smaller change sets with each release. It used to be that were lucky to get a new build per month. So you can imagine the number of changes that were in that build, and when you're on a monthly cycle, every release is an event. People gear up for it. They get ready for it. They stay up for it. They watch the graphs like a hawk, right? But when you're doing continuous delivery, when you're putting out new builds everyday with small change sets, it becomes less of an event. In fact, it's a non-event. "Oh, did we release today?" "Yeah, I guess we did." "I didn't really think about it. It just happened automatically [inaudible 00:47:19] machinery. If they had been a problem, we've got great alerting. That would have cut it. I would have known about it."

So it's a combination of these practices, continuous delivery. It's a combination of improving our monitoring, our alerting. It's also adapting practices like canaries. We use canaries through the Kayenta, our open source Canary system. It's part of Spinnaker, our great open source delivery tool. Through these great tools and these practices, they really got to a place where outages are a thing of the past.

[SPONSOR MESSAGE]

[00:48:02] JM: FindCollabs is a tool for managing hackathons and innovation within your company. FindCollabs allows anyone within the company to create new ideas and build

momentum around a new initiative. FindCollabs allows your smartest, most driven employees to build projects organically. If your company is looking for new ideas and innovations, check out FindCollabs. It's free and it was started by me. It's something I genuinely believe in. If you have any ideas or complaints or criticisms of FindCollabs, you can always email me, jeff@softwareengineeringdaily.com.

FindCollabs lets people within your company create new ideas. Whether you want to run a hackathon and generate new ideas or you want a long-term system to manage innovation within your company, check out FindCollabs at findcollabs.com.

[INTERVIEW CONTINUED]

[00:49:07] JM: The Netflix tooling teams I think are really a model for any of these large companies, like I think of large banks or large insurance companies that are trying to become software companies. In Netflix, you have these very niched tooling teams. So the edge observability team, for example, or the – I can't remember what Vasanth on the show to talk about the Netflix serverless-like platform. But that was a very niche set of developer tools that he's helped build within Netflix. How do these developer tools teams get spun up? How does Netflix decide what kind of niche developer tools are worth allocating resources towards?

[00:49:56] GB: Oh, that's a good question, because – And people often ask me, because on one hand, we want central teams that provide a tools that gives value to all of Netflix, every part of Netflix. On the other hand, edge engineering may have specific needs, and we need a tool to meet those needs. We have a model whereby we have central tool teams, like our delivery engineering team. They provide Spinnaker. We also have something that we call local central teams within edge engineering, and these local central teams will provide tools that are may be very specific to edge. A great one is a new tool we call Telltale, when there's a problem with streaming service, with the ability to playback a movie or something.

Where in the system did this breakdown? What is going on with the systems? How can we get sort of stirring a lot of graphs and looking through logs? How can we figure out what's going on? Telltale is looking a bunch of signals. It will alert us when things maybe start to go a little off the rails and it does correlation between, "All right. Your upstream and your downstream services

are showing this. You got two downstream services that are also having high latency and having problems. There's a correlation here. There's something going on." So this is a tool that is really designed to solve a need within edge.

We've got another great tool called Edgar for tracking. Streaming requests sounds simple, but they're actually very complex going through a number of layers of services. Where does the streaming request end up and how did this customer have a problem, and is it only affecting one particular device or one particular movie or one particular bit rate of stream?

Well, it used to be that somebody from, say, the Android team or the iPhone team would come to our edge developers and say, "Hey, can you help me troubleshoot this?" We'll sit down and trace this through the logs and figure out what's going on in each step of the system.

Well, our local central teams built a tool called Edgar and it will do a lot of that tracing for you and tell you exactly where this request is going or what's happening along the way. So in that way, it takes a lot of this sort of investigative toil off our edge developers and makes it self-service so someone can go and get their questions answered by themselves very directly in a clear way. So that's an example of a local central tool that's meeting a specific need, a specific pain point that edge developers might have had previously.

Now, at some point, yeah, some of these local central tools, maybe we'll find use cases for them outside of edge and maybe we'll make this as part of a central tool at some point. So there is a model for central tool teams that build these great, big tools, like Spinnaker, like Alice. Then there's a need for something a little more local, a little more close to you, and that's our local central teams.

[00:52:53] JM: What are you working on at Netflix these days?

[00:52:55] GB: Ah! So these days, one of the questions that people ask me, they say, "Greg, you've described this full cycle model where your developers do everything. They do code. They do testing. They deploy. They do operations. Why are you in SRE? Why does Netflix need SREs?" Good question.

I'd like to think of myself as a consulting SRE. I've lived through a lot of these stuff. I know what the pain points are. I can work with teams to find better ways to do that, to improve the tooling. Let me give you an example.

When I heard that the delivery team was going to build a new delivery tool. This was Spinnaker. I reached out to that team and I said, "Hey, I hear you building a new delivery tool. I've got these great use cases and these great pain points at edge. Here's what we're experiencing every day. Pleas, build us a tool that meets these things." We don't want a tool that does everything in the world. We just want a tool that meets these needs.

The Spinnaker team, they were eager to come and sit down and talk to edge engineers and find out those pain points and build the tool that met those needs. Now, our edge engineers use Spinnaker on a daily basis to deploy their own code and their own changes directly into production, then they feedback ideas to the Spinnaker team. Again, this is one of the examples of these partnerships. So I'm looking for these opportunities.

Another example; a few years back, I heard about these guys that were doing some work in – They were developing an algorithm for outlier detection. In this case, they were looking for anomalies within all these metrics that come off of our streaming clients. Is one particular type of device having trouble playing one particular movie at a certain time? Very, very needle on a haystack kind of things, right?

Our edge teams had a similar problem. Occasionally, we might run hundreds of instance, Amazon EC2 instances. Occasionally, one of those would go bad, and it will go bad in such a way that it would start spewing out errors just enough to maybe cross the threshold. A developer might get paged at 2AM, "You've got this error rate that just crossed a threshold. What's going on?" After an hour you trace it down to one bad instance. You kill the instance, error rate drops back to normal and everything is good. This is a really frustrating problem to deal with at 2AM.

I was reading through some newsletters and status reports put out by different teams. I saw this thing about these guys developing this algorithm for anomaly detection. I went to them and I said, "Hey, we've got a similar problem. We're looking for needles in a haystack of EC2 instances. Can you help us with that?"

So, together, we built a system which we called Kepler for finding these outliers. It would look through a bunch of metrics and, sure, there's an easy case where your bad instance is spewing out errors at 100%. That's the easy case to detect. What if your bad server is spewing out errors just above a normal range? Most of your instances are going to fall within in some range. What about the server that's constantly outside that range?

[inaudible 00:56:10] to detect, particularly as your traffic patterns change during the day, right? So how do you find those suspects, those needles on the haystack? Well, we put together this tool we called Kepler and we developed it. I had the use cases and the data. They had a great algorithm, and together we developed this tool.

This is a sort of thing I really enjoy working on these days, is I know the pain points. I know we can do better. I know we've got some really great tool teams. We've got some really smart people. Let's connect them altogether. Let's solve our use cases and pain points. Let's provide [inaudible 00:56:48] and iterate on them. At some point, I step out of it. This is now a tool that our developers can use. They can make their own use of it, and I'm on to the next thing.

[00:57:00] JM: So you are doing a lot of walking around, just serving the company, getting a feel for what's going on in the company, finding different issues, trying to see problems before they crop up.

[00:57:15] GB: Yeah. I don't know about the whole company, but at least within the streaming, I started in the streaming, well, what was then not streaming, but the movies over the internet. I started there. That's the domain I know well. I worked with these teams. I've seen them evolve. I've been there from the ground-up. I've built great relationships with the development teams, with the central tool teams. I know that people, we have mutual trust. We have mutual understanding. If I come to a central tool team, say, "Hey, we've got this problem in edge. We've got an issue. We think you can help." They're willing to listen.

So in this way, I do look for pain points. I do observe what's going on. I talk to our edge developers and I find out what the new pain points are. I try and connect with the tool teams. I

can help them. I try and maybe find existing tools and maybe this tool would almost meet our needs if we just added a few more feature. Let me help them shepherd it.

One of the things I find that's really exciting is we've got these great tool teams out there. They want to build these great tools. They just need use cases. They need to know what to build. They don't want to be guessing at what your needs might be. They're willing to listen. I come with pain points and use cases and show them how their tools can really help us. They're eager to get that out there. They want to build solutions. I have problems that need solutions.

[00:58:43] JM: Netflix is often a forerunner in seeing problems that the rest of the industry is going to encounter two or three years before the rest of the industry. What kinds of chronic or repeated issues in the developer tool space or the operation space are you seeing around Netflix today that you think might be harbingers of the rest of the industry?

[00:59:14] GB: You make it sound like this is sort of a deliberate search for the next thing. I wish I could say it was that deliberate. We just have pain points that come with our scale and our complexity.

[00:59:24] JM: Sure.

[00:59:26] GB: And we have people who are looking to solve these pain points. We have people who want to address them. We have people that say, "We can do this better. Let's take a bet on this. Let's figure out how we can find some solution for it. Maybe a tool doesn't exist. Fine, we'll build it."

So we're building tools that solve our pain points and our needs. Right now, one of the things I'm seeing is we've got a lot of great tools. They're very powerful tools. They can also be complex tools. It can also a lot of learning that has to go into using this tool. It can also be a question of, "All right, what's the most effective way to use this tool?"

[01:00:07] JM: This is like the conversation you and I are having the other night around documentation and employee onboarding and how do people learn to use all these stuff that

we've built within the company when they join and they're like totally confused by how anything works.

[01:00:23] GB: Yeah, and it's not only how do I use it most effectively. I don't necessarily want to spend my time learning how to configure a tool, a tool that's rapidly growing and changing. I don't want to use that tool to solve the problems I have. I want to use that tool to make my service more available to increase my velocity, to make my service do something even better.

So there is the onboarding, but also, these tools are changing and evolving. How does the developer keep up with all the latest changes? You could almost spend a lot of your time trying to keep up-to-date on things. Well, maybe there's another way forward. So we're just trying to think about the idea of managed solutions. Maybe we can build a layer on top of this tooling that will bring in opinions, that will bring in best practices. So a team doesn't have to try and figure out those best practices for themselves. They can be provided by a central team.

[01:01:24] JM: This idea of the full cycle developer that you gave a talk on, that you've been, I guess, communicating, evangelizing within Netflix. I feel like even at companies without the tooling of Netflix, even if you're just developing on AWS, for example. If I'm all in on AWS or I'm all in on Google Cloud, these cloud platforms are so robust. They provide you such high-level tools at this point that it seems like developers can be increasingly full cycle developers even at smaller startups just by virtue of the tools of the cloud provider. Do you agree with that?

[01:02:08] GB: Oh, I totally agree with that. This is an exciting time to be in where you no longer have to roll your own tools. You no longer have to start from the group-up building from scratch. There're a lot of great offerings out there. People say, "Well, we're not as big as Netflix. We don't have central tool teams. We don't have entire teams to be able to do really good tools, like Spinnaker. How can we do this?"

Well, there's lot of great open source tools. We've open source some of our own. There're a lot of great SaaS solutions that you can use. Even cloud providers are providing you with tools. You don't have to start from scratch anymore, and that's exciting. So part of what I'm trying to get across with this message of full cycle developer is you don't have to repeat this learning process that we went through. We try and multiple iterations. We try different things. We went through

this long, painful, years long process to where we are, and by no means it's the model, this full cycle model, the be all end all. We're still evaluating what's not working and how we can do this better? What are the new pain points? So this will continue to evolve. The important thing is we're willing to take that hard look at ourselves and say, "How can we do this better?"

[01:03:24] JM: Greg Burrell, thanks for coming on Software Engineering Daily.

[01:03:27] GB: Thank you.

[END OF INTERVIEW]

[01:03:31] JM: GoCD a continuous delivery tool created by ThoughtWorks. It's open source. It's free to use, and GoCD has all the features that you need for continuous delivery. You can model your deployment pipelines without installing any plugins. You can use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your cloud native project. With GoCD on Kubernetes, youdefine your build workflow. You let GoCD provision and scale your infrastructure on-the-fly, and GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn how you can get started.

GoCD was built with the learnings of the ThoughtWorks engineering team, and they have talked in such detail about building the product in previous episodes of Software Engineering Daily. ThoughtWorks was very early to the continuous delivery trend and they know about continues delivery as much as almost anybody in the industry.

It's great to always see continued progress on GoCD with new features like Kubernetes integrations so you know that you're investing in a continuous delivery tool that is built for the long-term. You can check it out yourself at gocc.org/sedaily.

[END]