EPISODE 14

[INTERVIEW]

[0:00:03.3] JM: Chuck Rossi, welcome to Software Engineering Daily.

[0:00:05.3] CR: Great, thanks. Glad to be here.

[0:00:08.7] JM: The last 14 years of your career have focused on release engineering. What happens during a software release?

[0:00:17.5] CR: Well, it can be a complicated process, it can be very simple process and I think the important thing is that you have people in place who know how to deal with the intricacies of getting software together to be shipped on hand. And competent people to do that. It's the last step before you, you know, all the work that you or your developers do to get what you created out the door has to go through some sort of release process to get wherever it's going to go.

I focus my career on kind of perfecting that and we can go into the details of what it means to put something together both for front end, back end, mobile, what have you.

[0:01:00.1] JM: How has that software release process changed over the last 20 years?

[0:01:06.4] CR: Quite a bit. Ironically enough, my first experience with this was as an intern in college. My college had a mandatory co-op program. I had to do a year of internships and my second one was with IBM, out in Denver, Connecticut and I kind of fell into the release role as an intern and let me tell you, that was, want to say, 1988.

The process changed a lot from those days. After that, I worked for IBM, I was a foot soldier in the great operating system wars of the late 1980s, early 1990s which turned into the Unix wars of the 1990s. I was the release engineer for part of IBM that was working with the open software foundation. Here you had a release process, we were building a rival version of Unix – to compete against the evil empire AT&T and System 5 and we were the BSD freedom fighters and we were going to – we were trying for operating system dominance back in those days. At least on the Unix platform.

That process, that release process involved me porting changes to Lipp C, packaging the whole Unix system from the kernel to the libraries to the utilities and getting that out for testing and evolution because these operating systems were being developed. This is a multi-month process, it was - if I made changes to Lipp C, I would see those changes in production and you know, maybe eight to 10 months. That's where I started from the release engineering kind of world and that was the old school, I guess you would call it the waterfall kind of date based kind of release schedule kind of thing.

Then, as I started to go into startups, that obviously didn't work, my first quasi - startup with Silicon Graphics where we're developing an interactive television system, I did more development work there but it's some released deployment stuff there. Here we have a more rapid system where it's a lot of prototyping and getting things in the hands of our partners and test customers. That still was weeks.

[0:03:23.4] JM: I think we could spend a lot of time on your background in release engineering, maybe we could do that at a later date or something, talking about your experience at VMWare as well. I think the release engineering process at VMWare when you were there from 1999 to 2004. I'm sure that's an interesting story to tell.

But, to fast-forward to the modern internet era, you were at Google from 2004 to 2008, you were one of the early release engineers there. What practices around release engineering did Google pioneer?

[0:04:01.6] CR: There was probably nothing formal but that was the first time I got exposed to a more rigid date based release system that worked pretty well. I was working with Adwords and Google Maps and some of those systems and releasing those. We had a kind of a two weeks cycle for releasing Adwords which worked relatively well. It was a very human intensive way to do things.

But that was my first exposure to a good kind of managed system with like good TPM's and PM's and engineering that was all on board with this process. Very solid source control system which was you know, perk force in those days but quite modified and [inaudible] version of perk force. Those concepts of that fixed cycle, the two week cycle kind of perfected was what I experienced at Google.

[0:04:57.7] JM: You joined Facebook in 2008. How did release engineering at Facebook differ from that previous job at Google?

[0:05:07.0] CR: Everything kind of changed when I went to Facebook. It's like, you kind of took all the rules and threw them away and started fresh. Maybe 300, 400 people when I started at Facebook, maybe a hundred or so engineers. I don't think – a very few of them have ever held a job before, including the CEO.

They were – all the founders were just out of school and they were winging it. They were doing what they needed to do to get this very scrappy, fast growing, really exciting place on its feet. There was no – I don't want to say there wasn't any thought to it but it was a very organic like, we're just going to get stuff going which meant you know, SCPing PHP files out to the servers, that was the release process at times.

I came in there and saw this and was faced with a decision. I had 20 years of experience basically coming in to Facebook at that point. I thought, okay, do I kind of lay down the law and go to the best practices I learned at places like VMWare and Google and IBM, wherever. Or do I kind of go with this organic new thing that they're working on here and let these crazy kids go and do what they want and hope for the best.

We pretty much did the second thing. The good news is the people there were really on top of this. They treated release engineering, the release processes as a first class citizen. Since the day I got there. That was important, it wasn't some like, after thought of okay, we need to have some build monkey, you know, pump this stuff out. Some of the best engineers in the company including like some of the founders were working on the release process and I learned the release process from you know, one of the founders.

That attitude of release being first class citizen, getting proper tooling, proper tension was what let us kind of go wild and invent and be very aggressive to build a release process that met the needs of this company and its growth.

[0:07:16.0] JM: How did the Facebook product compare to the Google products in terms of how it impacted the ideal release cycle?

[0:07:33.1] CR: The Google product, the most – you know, the crown jewel was Bliss, Google web search which had a very regimented release process. I didn’t touch that, that was a very kind of – a lot of SRE’s worked on that and it was a very delicate, prescribed, kind of piece of work. The Facebook release process, the crown jewel of that time was facebook.com.

It was a little bit the opposite, it was always in motion, always on the brink of disaster but always given a 100% attention but the speed at which we would do things was the main difference between kind of what I had experienced previously and what Facebook started to do and that really set the company culture for a mindset of you know, the move fast mindset.

Where we’re going to be able to move fast, things might not go well but we can fix them quickly, we can react quickly, we can develop quickly, we can do product release quickly. That’s the main difference I found from previous places to Facebook.

[0:08:42.3] JM: Did the Facebook code base have may tests when you joined?

[0:08:50.3] CR: Hard to say, when I joined, it was more organic. Facebook did buy into the whole idea of test your own development. There was again, at the core culture that you wrote, your unit test, you wrote your tests with your code and the tools, famously Fabricator, the tools were geared to help promote this. New people coming in and you know, experienced developers would always be flagged if they didn’t have tests with their code delivery.

The culture had some things built in to give it at least at the base level of your testing pyramid, kind of the bottom levels were well – had a good foundation. There was that culture of yes, the developer is responsible for the tests and this kind of – this is well before the 'dev ops' kind of model or mindset came along. We were organically – had this dev ops idea at least it’s kind of core tenant of developers are worrying about operational and testing and deployment things before it was kind of a thing.

[0:10:03.8] JM: There is a platonic ideal of a release process where you push some code and a fleet of unit tests run across that code and it verifies that this is not going to break the software build and then it advances to 1% of the audience and then to 2% of the audience and then gradually increases to 100% of the audience as the software increasingly gets deployed without showing any kinds of anomalies in the world.

There's also an optional procedure in that release process where perhaps somebody is actually manually testing things and clicking around and they are in that 1% of the audience that the code has been released to and they're just working with that code or working with that new software environment, that new deployment themselves, before it gets rolled out to the entire audience.

I think in many cases, the release process, the modern release process includes some amount of testing like a unit testing and then some amount of manual testing. Could you give me your perspective on how much that aligns with what you believe is the modern release process and specifically what you saw at Facebook?

[0:11:34.2] CR: Yeah, this is a sensitive area. I'll be – I'll give you my non-politically correct answers. Yeah, you're exactly right, there is that tension between the desire to manually test, to make anything perfect and go out. I think we're past that because your model is basically saying, we can deal with manual testing if it's asynchronous. I'm okay with that and this was a big fight at Facebook and I'm sure other places I've given talks to zillions of places about kind of this process and for the more established companies, the ones that have been around a long time.

I tell them, "If your release process depends on a manual step before you release, a manual test step before you release, your release process is fundamentally broken, okay? You cannot be agile. Anything, you cannot be continuous, whatever, you cannot do anything if you have manual testing part of your deployed regimen."

You know, that’s a problem for some people that can’t get their head past it. Facebook, yeah, we didn’t have a "QA group" to speak of, there was no – we had some contractors and some specialty QA people. I mean, Oculus is a famous example, right? You can ‘t – for a hardware device like the Rift or any of the Oculus devices, you fundamentally have to sit some poor soul on a chair and spin them around and see if they get sick or whatever it is, right?

We would only use kind of a manual test system or group for very special things like the product is too new and the unit tests aren’t fully there, the testing regimen has some special thing that we can’t do quite well automatically yet. We’ll patch it with some manual tests. But most likely, they’re going to be asynchronously done, they’re not going to be part of my release process where I’m going to wait for a test result to come back from any sort of manual testing.

That model, I’ve seen have success, at huge scale at Facebook. I mean, we were basically able to take the web front end which has thousands of changes a day and move it from a one week release process to a three times a day release process to a quasi continuous releases process by the time I left. All without synchronous manual testing, like you say. It’s more about having teams be able to test it in the canary phase along with the other canary tier.

[0:14:02.8] JM: Describe the release process when you joined, in a little more detail. What was the cadence, what were the bottle necks, what did a release look like?

[0:14:14.9] CR: Facebook came up with this way of releasing and it’s a branch cherry pick model and I’ve used it in other places to some degree but Facebook did it at a degree that was pretty huge. The branch cherry pick model, I really like and at a certain scale from zero to maybe a thousand developers, I would advocate for this for kind of front end or even back end release.

The model is basically, you make a weekly release branch and once a week is about as long as you possibly want to do that and for Facebook, for whatever random reason, it was Sunday at 6 PM, we would cut a release branch using whatever Adobe source control system, you can tolerate.

After that branch is cut, you only take cherry picks for a master and we had a tool that allowed developers after they checked in the master and say like, "Hey, I don't want to wait till next Sunday to have this in the next week's release, this needs to go out before then." That left it mostly up to the developers or PM's or whoever to decide if the risk was worth it.

But here's the deal. My release engineers or me personally to begin with, but then my releases engineers and I would make the final call if we're going to take that cherry pick because those cherry picks would go out once a day. The weekly release would go out basically at a Tuesday. That was about, in the end it was about 10,000 diffs.

Like a bundled up in that weekly cut that was tested from the branch code on Sunday and released on Tuesday and then after Tuesday, on Wednesday, Thursday, Friday, Monday, we would do this cherry picks and we took about two to 400 cherry picks a day. But almost all of them were vetted to some degree by a release engineer or an eventually by a ride along engineer as well.

That process really worked well for quite a while and it was only when it started to fall over, we were exceeding a thousand diffs a day, we were exceeding four or five, 600 cherry picks a day. We realized, the number of cherry picks and then the commits are converging.

That's when we made the announcement that we were going to go to a quasi-continuous front end release and that's what Facebook does now. Basically, there are no more branch cuts, no more cherry pick model. You commit the master and master ships every hour or so, every two hours.

[0:16:40.1] JM: Facebook has a system called push karma or at least it used to have that system. Can you explain what push karma is?

[0:16:48.0] CR: Sure that goes along with the cherry pick system and I'm going to be clear, there's a lot of subjectivity that goes into deciding which picks to take. I mean, there's objective things you can do as far as metrics to understand that the code is safe but there is a subjective component that is critical and then again, are the humans involved and when I took cherry pick and it went disastrously wrong, it's fine.

I mean, the key was the person who committed the code was around to help me out and those days, we were using IRC to coordinate when these pushes went out, when cherry picks were taken, I would only take a cherry pick if you were in IRC. We had all these tools in IRC that would say I'd push a button and they're be like hey, we're going to start this daily push and take this cherry picks and would call out the developers by name and in IRC which would alert them in their desktop.

If they said, if they responded in some way and saying, "I'm here, I'm alive, I'm breathing." Their cherry pick request wasn't too risky which the majority of them were, we would take it. If they didn't respond, we didn't take the cherry pick so if you're not here, I'm not going to take your cherry pick because it's going to go live in an hour.

If they did respond, we took the cherry pick and it all exploded very poorly and we have a bad experience, that's fine but if we all did that and they're nowhere to be found and I'm getting no support and they basically abandon the release process, I'd remember that. I had my own internal database of like yeah, this guy, this gal was missing in action like multiple times. When it got to so many developers, I couldn't keep them all in my head, I made like this little secret star system on the cherry pick page.

Next to every developer was like, they all were born with four stars to their name. If a bad thing happened like that, we would basically give them the dislike button and they would lose half a star. We would log that, a little window would come up and we type in like what happened, "Hey, this is really bad, you were nowhere to be found, we had to revert the whole thing, I hate you."

That would go, we cooked some in and we go to the developer, it would go to the developer's manager and it would go to the kind of the performance review tool. This was very much a private shaming, a private discussion between us and its intent was to basically make us more aware, make us all more cognizant of the gravity of the situation and the dependence that we have for developers to be the backbone of this.

When they're missing and things go wrong, that's you know, that should be corrected. Hopefully that's not too crazy, no one ever got fired, it was mostly done in fun, don't over analyze it, overthink it but I don't know what your thoughts are in that system but that's what we did for a while.

[0:19:38.3] JM: Well, it sounds very useful, there's this tension in kind of the social networkification of the enterprise where in social networks, people develop their reputation and the reputation is mostly implicit, right? It's usually not explicit. You know, unless we're talking about like Yelp businesses or there was that company, I think it was called like People for a while where it's like Yelp for people basically and people just hated it so much, they just were appalled by the idea that I think the company shut down or it had to shut down or something like that.

People are very anxious about this idea of being kind of rated in a semi-public fashion but we know that crowd sourcing works, so it is this tension between utility and social health, I suppose.

[0:20:40.4] CR: Yeah I agree, my intent was not to embarrass anyone or not cause anyone trouble, this was all done privately so no one could see anyone else's kind of rating or whatever. That would have been a much bigger HR issue if I attend that. We were careful. I was very aware of kind of those issues that you've raised and we never got too carried away, it was mostly done in fun but you can see, this is interesting.

The system, we didn't really, we started not using it as much, that kind of karma system but when we did, when we went to continue the deployment was you know, no podcast would be complete if I didn't mention AI or ML or something. I won't mention it. We had very rudimentary kind of ML. Think about this, on commits, there's all these meta data you have of who did it, when they did it, what did they touch, what files did they touch, I'm sure there's papers analysis on this. We wrote like just some simple kind of analysis of all these factors to come up with a diff risk profile.

For the continuous system where there is no chance to kind of look as things are coming through, we wanted some like little gross gauge on is there more risk here than not. Again, there was some things you wouldn't expect and some things you did expect when we ran that system.

It's still you know, kind of being developed and it's kind of being figured out but I really think there is some data in there and that metadata of when, how, where, what code, what code constructs, you know, there's so much data on – we have a very good said review process when things go wrong, you know? Call this up.

We go back and analyze it, that data goes back in the system so we know which files were touched, we know who touched it when, how and where and even where they're. There are some interesting things that's been done there. I would like to explore that some more.

[0:22:35.5] JM: Right, so you're saying like, if there is a line of code, perhaps that writes to a core database, you know, that detects what your – who your friends are or what the things in your newsfeed are going to be. This might be very sensitive lines of code and it will be useful if there was some metadata analysis of what kinds of lines of code or what specific lines of code have triggered high severity outages and high severity situations?

[0:23:16.2] CR: Exactly, you could imagine looking at it like – you can bring up a C++ file and you can see a heat map of like where risk is versus where it hasn't been historically.

[0:23:29.7] JM: To come back to the social aspect of how company – internal company dynamics work, when Nick Schrock came on, he talked about this other social like phenomenon within Facebook. The idea of the influencer engineer. Basically, the idea that there were certain engineers that within Facebook, if they had an idea, they would marshal resources behind it.

Whether or not other people came along for the ride and they would build a reputation as somebody who can get stuff done and get stuff off the ground and eventually, when they started doing – when this person would develop such a strong positive reputation, if they did something new, people would just gravitate towards them and I just mentioned this as another phenomenon in which the internal Facebook engineering culture resembles the social networking world, that Facebook has contributed so significantly to.

I'm wondering if that resonates with you and if you have any other examples of ways in which Facebook embodies the world of social networking in terms of its engineering processes?

[0:24:56.9] CR: Yeah, I know Nick is talking about there and you know, there were definitely engineers who had that persona and could rally things and really get tremendous leaps. I guess sometimes you call them the 10X engineer, right? But with an added benefit of being more of a leader or able to convince people to come on board. They are like the people who are a curse and a blessing.

I have many samples, I probably can't get into details and rent a name but there were certainly people who I want to absolutely strangle at times because they literally took down the site with their shenanigans and at the same time, I know the same person was the – a week ago, single handedly brought the site back up when there was no hope of any of us understanding like what just happened, right?

There were dozens of examples of these people who are like, that guy again, I can't stand it, how did he manage to do that and then a week later like, gives you this amazing tool that you know, saves the day. That is a challenge for organizations and like you say, you have a kind of a social strata that mimics some of the things that we see in social networks. Certainly within Facebook and it is if it would be more of talk about how to be a good manager is you need to manage those clicks and manage those people because sometimes, in Facebook.

Another series of examples of people who were very passionate about going in a certain direction. This is that any company actually I saw this other companies and they're just not going in the direction that's great for the company or for the core mission that's more – some are religious belief to go in a certain way and things like that can be problematic and that's where you have a little bit more conflict of personalities and people will go way mad and you might lose some folk who kind of religiously want to go a certain way but management or the core mission doesn't align with that.

[0:27:00.2] JM: Continuous delivery works pretty well for web applications and back end server applications. In the mobile environment, you have a release process that is gated by Google and Apple. There is a process by which you as a mobile app developer have to release an app to the review process and then apple or Google has a review process that they invoke on your mobile application. This can be frustrating for mobile developers that want to move fast.

It really slows down the mobile release process in some ways. How did the mobile release process work at Facebook?

[0:27:53.3] CR: Okay, you want to go there, you don't want to let me off on this one?

[0:27:56.7] JM: Yes.

[0:27:58.6] CR: Okay. Now that I’m not encumbered by being employed by anyone, I can get to the real nitty gritty here. I cannot express. I worked in the space for 20 years, 30 years and basically when Facebook went to mobile, we took the last 10 years of all the best things we did about release and all the different companies and all the different best practices and all the release things we did and all the improvements been made on how to do things safely and correctly for customers and developers.

We took all that and we threw it away when mobile came. It was like going back in time, being, have ridiculous handcuffs placed on you and it was a nightmare. It is my least favorite thing and I defy you to find one mobile developer who will say anything good about the process. The process was born with the concept of like hey, there’s probably going to be like a hundred apps and they’re going to update every six months.

That was mainly how the stores work, to this day, the mindset of specifically iOS is 100% geared for that model to this day. We had a tough time so it’s – you're going from a world where our front end and back end release which was our world, we had total control, we could release every 30 seconds or every 30 days, we could figure it out, we could do it safely, correctly, what made sense for our business and to get it done. We went to mobile, we went to this world of like old style enterprise release.

It was even worse because I had four or five of the top 10 apps that we were releasing. I had the biggest user base of mobile users in the world. You know, across all those apps. It was the most terrifying thing to take 10,000 diffs, package it into effectively a bullet, fire that bullet at the horizon and that bullet, once it leaves the barrel, it’s gone. I cannot get it back and it flies flat and true with no friction and no gravity till the heat death of the universe. It’s gone. I can’t fix it.

That fundamental model is terrifying in effect. I first release on iOS after I got back from – I took a little break back, I forget 2016 or 14. Ig to back and the release process where I was, was going like hey, you know, let me take this over now that I'm back. The first release I did, I released the wrong app icon for the app. How many hundreds of millions of phones got Facebook with this designer holder for the app icon, I guarantee you, there is still a phone out there from 2016 that has this app on it, right?

There's all these new things you need to worry about. Jocelyn kind of got into this when you talked to her about the Facebook mobile release process. Ultimately, let me just make the statement like all mobile companies should be shooting for a one week release cycle on mobile. Google and Apple are not at all geared for that but your company is. Your survival, your core product needs to release every week on mobile and Facebook got to a one week release cycle years ago basically for all our apps.

That gives you the ability to keep your app alive and moving and keep your sanity as you develop your things. It is not going to make Google and Apple happy but that's not why you are in business. So I can get into a lot of details about what's wrong with the mobile release process, how to fix it. We tried to fix it many times internally at Facebook because we have the size to do it and there were epic battles between us and Apple and Google that kept us from doing the things that need to be done.

[0:31:51.9] JM: I believe these battles can be epitomized by React Native, which React Native allowed people to have dynamic, highly dynamic code in their mobile applications whereas in the world where you are just shipping an IOS binary that is written in Swift or Objective C, the review process can catch everything in the app as it has been written in that binary as if it has been etched in stone but when you create a dynamic JavaScript module that you put into your application.

You can deploy code dynamically to your users, so you can sidestep the etching in stone process that the android Java environment or the Apple swift environment imposes upon you. Did the creation of React Native create some tension between Facebook and Apple?

[0:33:08.6] CR: So you hit the nail right on the head there and I cannot express enough to people listening that things like react native or any bundled kind of over the air delivery is absolutely critical to your survival, to your app and I was a huge proponent of React Native at Facebook because it finally gave us the flexibility and the power to actually do what is best for our users and our process and our product. So yeah, we worked on that a lot.

There is always a tension more than I expected between the native developers and the ones who don't want to do native development or want to do the more React kind of based things. So that was one problem we had to get by. Now, to be really clear you are not going to get away from doing native code for all the low level and performing things you need to worry about but there should be an attitude in the company, any mobile company that I'm like, "I am going to push as much as possible into some sort of bundled delivery system."

The other thing that was huge in the space was having a gatekeeper system or a feature flag system. So we have a famous gatekeeper system both for front end, back end and mobile that these future facts can be changed via configs on the fly and updates are pushed out to the client devices that will turn on or off features, which is huge in letting you manage how things are released, when they're released, to whom they're released and if there is problems, how to instantly turn off things.

If you are having these flags available to say, "Oh, that functionality is pretty broken right now. Let's turn that off globally." Without having to do a hot fix, without having to go to the store, without having to wait 24 or 48 or 72 hours to reach your customers. So yeah, I cannot stress enough that and again, I like React Native a lot but there's got to be something that gives you this control back to your company.

[0:35:15.6] JM: But did it create – I mean you don't have to answer if you don't want to but did it create any tension between Apple and Facebook? Because you are kind of taking away their ability to review the apps.

[0:35:30.2] CR: So the Apple threat is real even after I left and now that I have been out of Facebook for a while. It is a very real thing that they will retaliate if we go too deep into this. They might even retaliate against you and your podcast, you'll never know. So yes, there was tension and like I said, there were epic battles fought, won and lost but yeah, you are going to take some risks with again, Apple could come after you and things might be tense but I will maintain.

You should do what is best for your company and for your users. You are not there to please Apple, you are there to do your mission, do your vision, get your product out there and do your thing. There are horror stories out there. I have heard them from any company that competes with Apple for certain app space things that are shenanigans that go on are just criminal. So yeah, I am not going to get too deep into but it's there.

Maybe someday I will write my book and then you will hear but it is a real thing. So you have to manage that but do not jerk from it like do what's best.

[0:36:41.2] JM: Yeah, I think for people who want to more about this in detail, they can just look at the case of Spotify and just read about what is happening with Spotify. Anyway, shifting to engineering, describe how Facebook did version control when you were there.

[0:36:58.4] CR: So we started, I think we were on Subversion when I got there and again, this was all – there wasn't a ton of thought put into it. It was like, "Hey, we got to do something that works," and this is truly whatever your company should do. Don't overthink it, don't plan for the next 20 years, get something going to keep your company going and alive that is not going to damage you. So Subversion for a while, we made the change over –

We never went like full Git. I am not a huge fan of Git at least in the applications that I need to do things, which are big companies with huge monolithic codebases, which was both Google and Facebook. So just to give you an idea like all of the Facebook frontend was one repository. All of IOS was one repository, all of android is one repository, a repository more or less and there's good – you know you've had discussions about this and we could debate the merits of multiple repos.

And then a single giant repo. I am a big fan of the giant repos. The tools of where that falls over. So we took up basically makingMercurial be the winner in the space for anyone who wants to do things at big scale and the reason why was Mercurial is still a living thing. You can still modify it, it has the idea of modules is written in Python. You know Git is a stream of consciousness C-code from Linux from like 20 years ago, right? So you are not going to touch it.

You are not going to improve it. You can't really do anything with Git. It is at its terminus. So Mercurial has the same concept. It is very mappable, one or the other. I mean if you are a developer, don't get religious about what, you know make your company work. Don't get crazy religious about what it is your views and get Mercurial or whatever. If it is not fitting your needs just say so and we had a team, basic developer infrastructure team that spun off.

So the source control team spun off from my team and we basically stood up a Mercurial team that very actively develops Mercurial in open source land. We don't have our own fork or whatever. We basically use what is out there. That is our – we are technically coupled to that. So in the end, we just did kind of round someone up here. We have monolithic trees managed by Mercurial with our own set of tools and special build processes that optimize the process to deal with things that are scaled.

[0:39:36.9] JM: Why don't more companies in the industry have – actually I guess I should just say briefly, Facebook has a monolithic code repository as does Google and a monolithic code repository differs starkly from how most companies in the industry manage their repositories where they have a wide variety of repositories and the code just gets deployed and merged independently and there is no really an overall unification of the code base.

Why don't more companies in the industry do the mono repo approach that Google and Facebook have used?

[0:40:16.2] CR: I think there's issues with tooling support. I never tried to do a mono repo without having a big team around me. So I wonder, I think it would be better now with the improvements we have made to Mercurial that you could decide, "Okay, we're going to put this all into one giant repo and there is enough tooling support out there that we can make this work." The problem is it goes beyond just the source control system and your built system has to play nice with that.

Even the bug system and any sort or system that needs to go into the source repository and do things to get metadata or to get change history or what have you has to be big repo aware. So we should do more to improve the tooling for starters. So that would make the water a bit more warm for people to get in. The other one is I imagine the problem is I get or if you are using GitHub, it is going to push you towards the smaller repos, the multiple repos situation.

So you are getting biased right out of the gate if you are stuck on GitHub or whatever and that I don't see a great solution because if you are going to use Git, you are not going to use mono repos. So I don't think that someone is doing some work that I am not aware of that's going to make it work. So I would love if Google and Facebook even pretty good. I would say Facebook especially but about giving as much tooling away as possible to make this work.

The problem is that so much of it is specialized and you know the story, it's hard to generalize enough and the effort it will take to maintain it in a good state and general enough that we can put it out there is non-trivial. So it is a commitment for places like Amazon, Google and Apple to – sorry, Facebook to get that, make that tooling available.

[0:42:04.8] JM: What did Facebook build in terms of continuous integration tools?

[0:42:09.6] CR: So we again have this developer infra, this dev infra team that builds everything that developers need. I am talking – so I was into that team for heading up release engineering part of it. So the release tools and the cherry picked tools and the whole continuous process and all of that, we did that but then you had teams providing developers with an IDE. So you know we have the atom based nuclei IDE that we worked on and then open sourced language.

So all the PHP iterations of hip-hop and HHVM and all that came from us. All of the JavaScript language improvements we did all came from this team. The build system buck came out from these teams as well. So we've write the build system, the integration with the fabricator or the diff tool and the source browser that's built in there as well. So all these tools are born from this group and all available for the developer and this team's mission is to make developers happy.

So they are constantly running surveys and running metrics and have dashboards up and down the block on how long does it take them to build these certain things. How long are developers waiting? How long are the tests taking to run? How long are the builds taking to get out? And these teams are dedicated to making that as small as possible and Google has the same concept, same team idea. So that is the ultimate expression of developer productivity and the tools that can be provided.

And again, the good news is that a lot of this is put out there. You can crib a lot of this for yourself for your company, for your startup and if you think you're going to scale huge, you can. I like the idea that if I am going to build a company and I am worried about scaling, I just look to places like Google and Facebook to say, "Okay, they solved it." They tell you how they solved it. They sometimes provide the tools to solve it. I am going to take advantage of that. So it is great that it's out there, so I was very happy to be a part of that.

[0:44:25.4] JM: Do you have a good anecdote of a time when you approved a release, the release went out, things broke, it was catastrophic and then it was very hard to roll back? I'd like to get a perspective for the firefighting process as you experienced it in Facebook.

[0:44:53.5] CR: So we had a lot of fires and there's plenty of – you know it is a sensitive area. I don't want to get too detailed into how things went poorly. I think the important thing to remember here is how you deal with these kind of issues. So yes, I can remember a time when someone made a change and it basically defeated gatekeeper. So all the gatekeeper checks and gatekeeper again, this is a feature for lack of concept like there's code on your phone right now.

Or a code in your browser right now that isn't being run and if you are a Facebook employee then you will see it. If you're not, you won't see it, things like that. There's six months of development in the app right now that you are not seeing. Some things will never see the light of day, some are going to be great new features. They are going to be launched in the future. Imagine if that check system just went away and now all the code runs, right? So that situation happened way, way back a long time ago and suddenly –

[0:45:59.2] JM: All the code?

[0:46:00.6] CR: All the code, run all the code everywhere for everyone. So the next six months of every feature ever to be launched was suddenly live and the reason we saw this was this is back in the younger days and we had this more of a rudimentary sight metrics and things. We just saw this crazy usage spike and network spike as all of these things turned on. It started shipping more data and people were using. It's like, "Why are we 15% higher at this time of day than we're supposed to be?"

And we suddenly realized it like there's no filter, there's no checks. They are saying everything out loud and we're in a panic because at that point, we didn't understand like, "Well what do we do?" I can get a fix together and push it out but it will take time to do the build and get it pushed to the fleet. It might be an hour and we can't have every secret out in the world for an hour. So the VP, I remember specially went and logged into the GLB and turned it off.

So we just turned off the global load balancer. We basically just turned off Facebook at the point of network, right? And we just shut down the site and regrouped and so, "Okay what happened? Let's go to the history, who broke it?" Not that it was an important thing but to just understand what happened and fix it and I pushed out the fix and we turned on the GLB and luckily in those days it was probably under a 100 million users that it just went not a huge deal.

So that anecdote was kind of towards the beginning of when we had a more formal post-mortem process and this was our SEV review process and the idea here is okay, let's not let these crisis go to waste. These anecdotes are great but what do we do after the fact is what's important. So we got all the department heads into a room once a week and we'd go over that week. We had SEV one, SEV two, SEV three, SEV one being the highest.

Let's go through the SEV one's and SEV two's at least and that turned into a more formal severe your process where we would do this post-mortem. The main thing is don't assign blame. We're not here to do which on or to point fingers or say this team blew it or this person blew it or whatever. It's like what happened, when did it happen, why did it happen and what do we have to do to make sure it doesn't happen again?

Log that, create the tasks necessary. If there is more of a meta issue about our alerting system isn't good enough or logging isn't granular enough or our manual system for testing or for watching things is not good enough, let us resolve that root issue and track it. So yeah, that was the benefit of all of these anecdotes or all of these kind of horror stories was that they got fewer and fewer and fewer because we'd never let one go by without picking it apart and improving on your length system.

[0:48:57.9] JM: The Google culture of incident response and release management, I think this is embodied in the SRE role, the site reliability engineering role. How does the SRE culture from the Google compare to whatever the equivalent was at Facebook?

[0:49:25.1] CR: So Google really pioneered that SRE concept, that site reliability engineer back when I was there and it is a special role. It's like half network engineer, half assist admin, half programmer and as you can tell, there's too many halves to put into one person. So it certainly started off as a unicorn position and the SRE's I have known some are legendary in their ability to solve their problems on the fly but the concept of the role was born there at Google and it certainly spread in the industry.

And people call it different things and the role is kind of a bit softer now as far as what it actually means, it depends on the company. We had production engineers at Facebook that changed the role a little bit. So that role is very much – they were always our close cousins in the release engineering world. So these folks have a tight coupling between the release people, the product people of what they're supporting and the ultimate responsibility for the site itself.

That is the role that I understood and I have seen it both at Google and Facebook that it is a really critical role. You need the right people for it. There is a fair amount of burnout and a little bit of drama in there because it is a more stressful situation because ultimately, these people are their first responsibility is to the site. Is the site up, is it healthy? Are the mobile apps up, are they healthy? Is the company's core mission are we serving?

And then if they are not, they're good at both debugging the problem and then figuring out how to put things in motion or at least alert the right people to get the response ecosystem up and running because you don't leave it to these people by themselves obviously right? They need to know how to alert, how to properly get the right resources in place and start spinning up a response team when things go significantly wrong.

[0:51:28.9] JM: One focus of this series of episodes about Facebook is to explore the myth that Facebook engineering was largely a replica of what had worked well at Google. My perception is that there are many ways in which Facebook contrasts sharply without Google engineering works. How would you compare and contrast the two engineering cultures of Google and Facebook?

[0:52:01.8] CR: Sure, there is many people who had this experience, a lot of people. I was one of the early people who went from Google to Facebook but there has been probably thousands who have gone from Google to Facebook and sometimes back. There are absolutely differences. They are completely different environments, completely different companies. So in the beginning people did incorrectly assume that we were a copy.

That Facebook was a copy of the Google environment and for sure we took some of their ideas and some of the best practices but Facebook had very much its own culture organically. I'd say the differences were definitely around speed, around execution so Google really dominated the world pretty quickly and had this leadership position and its technical excellence for quite a while. Facebook was a much scrappier place because they were fighting to establish a new space.

Although there had been social networks around but not at this scale and the growth rate and the amount of land that we were conquering, necessitated a much different culture to be able to move fast. To be able to make changes on a dime. Google didn't have that when I was there at least and I don't think it would have gotten better since I have left. So that was the big change. The other one is we were much more open about open sourcing, about talking about what we did.

I've talked for years about all the details of how we ship and how the tools that we use and all the secret sauce whereas Google was not open about sharing that or open sourcing things to the extent that Facebook did that the ultimate example of that was the hardware right? So Google pioneered the idea of like building your own servers. Like why do we need servers from Dell and HP with USB ports and all of these peripheral things and things we don't need.

A CPU, I need a CPU, I need disk, I need memory and network and go. So they did that and if you so much took a picture of the rack that those things sat then you are fired. You are done. Facebook took the exact opposite approach and said, "Yes we want to build our own servers. We're going to open source the entire hardware, our architecture, the racks, the ACDC, all of the converters, the electrical. We are going to do something innovative in how to deliver power efficiently to the power supplies."

And all of these ground breaking stuff and it was all done right in the open. So that's an example of the cultural differences between the teams. Engineering wise, you had top engineers inventing languages and better language constructs and better tools. That was similar and Google eventually was a bit more open about kind of having their tools cut loose and I think lately they have been a lot better about that. So there are some similarities there but fundamentally yeah, different environments when it gets down to some of the engineering details.

[0:55:06.2] JM: What do you miss about working with Facebook?

[0:55:08.8] CR: I mean everyone will say the people. I will say some of the people. It was a very, it is just to be surrounded by so much ability to solve problems was really what made you go back and keep working. There were things I did not think that were solvable that we solve and that was exciting. Also the thing I miss is being in the center or what's happening in the online world. I mean for better or for worse, Facebook and Instagram.

What's that messenger? The whole Oculus, the whole family of things going on there very relevant to almost everything in online life and to be the person with your finger on the pulse of that and my role of releasing that and making sure that all of these things got out and we're successful and could be updates and kept alive and healthy. I mean software is a living thing right? It's not that the folks who wrote CandyCrush wrote it and ship it to IOS and said, "Okay, we're done. Let's go get lunch."

Software lives and breathes and like my role is to keep this software living and breathing and getting that stuff moving and rolling, getting the customers. You know I miss being in the middle of that and you know, providing that conduit to make sure it's done safely, correctly and into the benefit of the customers.

[0:56:30.4] JM: You mentioned there were problems that you didn't think you would be able to solve that you were able to solve, do you have any examples of those?

[0:56:37.3] CR: I mean so this scaling stuff of how to serve billions of request a second, I am not as versed in the details of that. I just saw the end result. Like I said, I was in the teams that did some of that infrastructure in the backend and I was amazed. You should have those people on to talk about how they solved like cache issues and database issues and all of these things at scale. It's fantastic. The one for me that surprised me was the move to a Quasi container deployment for the web frontend.

I didn't think – I mean I thought we could do it but we did it in one year. I announced it on April 1st, 2016, which everyone thought it was a joke like yeah, we are going to go continuous deployment, you know?

[0:57:19.4] JM: Bad timing.

[0:57:20.4] CR: Bad timing, no I am serious. It was the opposite of what people expected. They're like, "Hey," I let it off and they're like, "Hey, we're getting so big and again, I am taking a thousand to 1,200 diffs a day, 10, 12, 14,000 diffs a week. We can't go at this scale."

So your normal thing is like, "Oh man we have to slow down," and I'm like, "You know what we're going to do, we're going to go faster." And so that was something I didn't think would be as doable as it was and we did it in one year exactly on April 4th of 20 – we announced it at 2016.

April 4th, 2017 we turned the crank to go to a 100%. We turned the fleet over to a 100% continuous, quasi continuous deployment in one year and that was something that at least for my team, it was personally very satisfying and something I didn't think it could be done. I knew it could be done, I wouldn't have said it if I didn't but you have your doubts, right?

[0:58:17.5] JM: All right, last question. What are some engineering lessons that other companies could take away from Facebook?

[0:58:25.6] CR: So I will focus on the plumbing parts because that's what I was most associated with. The first one and I eluded to this is make your release process a first class citizen. Give the right tools, the right budget, the right people to make this part of your culture and part of how you do things. Specifically, keep your release processes as quick as possible. If you're doing mobile frontend, backend, the idea of small manageable, quick releases has proved to be quite scalable and quite beneficial.

So again, move to a very agile of small discrete changes that are much easier to review, much easier to manage and to talk about mobile specifically, get to a one week release. Do this things as forcing functions. If you don't think you can, try because what you will do in the process is you will flush out all the things that need to be fixed and this is what we learned from the continuous deployment move is when we made this effort and had people working towards it.

All the things that weren't working well were quickly discovered and we could put resources in place to say, "Okay, we can't get to a continuous process because the translation systems can't do it that fast." Or, "This logging system can't keep up or this that XYZ." It is an amazing forcing function when you make a decision to do something like this. Be it one week mobile releases, quasi continuum web, whatever it is you can do, catch and release from four weeks to two weeks and see where that gets you.

And I guess the final bit of advice is have fun with it. I enjoyed my time, I think people enjoyed working with me because I was serious and dedicated but I also had a good attitude of like, "Hey, we need to make this fun and I don't want to work in a sweat shop." I work long hours because I enjoy it. I enjoy the people I am with, make it a fun process and get it done.

[1:00:29.2] JM: Chuck Rossi, thank you for coming on the show, it has been really fun talking to you.

[1:00:32.5] CR: Thanks Jeff, really enjoyed it.

[END]