EPISODE 02

[INTRODUCTION]

[0:00:00.3] JM: Jocelyn Goldfein, thanks for coming on Software Engineering Daily. You are a former engineering director at Facebook. You're today a managing director at Zetta Venture Partners. Thanks for coming on.

[0:00:11.8] JG: My pleasure. Thanks for inviting me.

[0:00:13.6] JM: You joined Facebook in 2010 as an engineering director. What was Facebook's process for senior recruiting at that time?

[0:00:22.2] JG: Oh, it was entirely ad hoc. I think I had probably 16 interviews across. I'm trying to think. I had lunch with the CTO, or I guess at that time the VP of engineering of Facebook, Mike Schroepfer. He persuaded me to start chatting with them. It really was just like, "Well, why don't you have coffee with this guy? Oh, that coffee went well? Have coffee with this other guy."

I think it was quite a few coffees before they and I were mutually comfortable to proceed to something like an on-site interview day, where I actually came in and met four or five people, including Mark. I think they may even have been a few follow-ups after that. I mean, Facebook was I think correctly extremely protective of their culture, and correctly aware that bringing in leaders, that senior leaders have the ability to shape and to warp your culture. If you're going to bring someone in at a senior level, you'd better be really, really sure about the fit.

For the most part, Facebook didn't hire managers from the outside the – I'd say 90% of the management team was homegrown. They were actually more prone to hiring senior leadership, because it would just take it from the outside, rather than frontline managers, because it would just take too long to develop senior leadership internally. I think that Facebook did benefit from some cross-pollination of ideas from other places, which I think – but we were the one vector for doing that.

[0:01:42.6] JM: What were the products that you were working on when you joined?

[0:01:46.5] JG: When I first joined, I was in boot camp like every other new engineering hire at Facebook. I spent my first six weeks fixing bugs, writing small features, pushed code to the site in I think my first week, or maybe the Monday of my second week. I can't remember. My first teams outside of boot camp were, I want to say search was the very first project. Then I quickly picked up newsfeed. I would say that my first major product milestone at Facebook came about a year in when we shipped the – what will be remembered about that release is that it was the machine learning rancor for the newsfeed. This would be in the fall of 2011, but it was also a total UX rewrite. We completely revamped the homepage of the most visited URL on the planet. That took a little bit of hubris, especially for me as a brand new to the company. That was an exciting project. Not well received by the user base of course.

[0:02:47.3] JM: It wasn't with the Facebook newsfeed, or the new ranking system? Oh, because they move from chronological.

[0:02:53.1] JG: The feed, I think originally shipped in 2007 or 8 before my time at Facebook. It was not the 1.0 at the newsfeed and was universally reviled. I mean, I think a million people joined a protest movement against the newsfeed in the first place, but eventually folks came around and came to love it. When you're adding so many users, the ones who joined post newsfeed, for them that's what Facebook was.

When we shipped the update in 2011, if your listeners have been Facebook members for long enough, they may recall that the homepage used to have two tabs; top news and most recent. Most recent was a chronological tab. Top news was a ranked version of the feed that tried to put the most important posts first. The importance was determined by essentially a rules based to algorithm. It wasn't machine learned at all.

We both replaced the hand-tuned rules based algorithm with a machine learned rancor and we more or less did away with the most recent, with the chronological version. We actually tried in a number of ways to weave chronology in with the rancor, but it was a little bit subtle and not obvious. In the end, we did also add a very – a somewhat hidden way to reorder your feed chronologically. I think that's gone now. We ran test after test after test. The chronological feed does not perform.

It's logical when you consider human nature, which is that humans give a disproportionate amount of time to the top 10 items in their feed. Those are disproportionately more likely to be seen than any other posts. Of the top 10, a disproportionate amount of that time, half of it goes to the very first post, the number one spot. Those 10 spots are gold if you want to drive reach, or distribution, or engagement, by the way, if you want to create value for the people reading their feed.

There is no world in which putting a story of random quality into the top spot is better than putting your best guess at the best story into the top spot and then to the top 10 spots. Even an imperfect rancor is infinitely better than random quality, which is what chronological amounts to. I meet lots of – my own friends and family will tell me confidently, "Oh, I'm sure that's true for the average Facebook user," but I read my entire feed and I require it to be in chronological order and I would read more stories and I would engage more and I would give more clicks, likes and comments to my friends if I had a chronologically ordered feed. and all I can tell them is that their human perception is wrong, because we know who the people are who like to read chronologically. They're the ones who bothered to go find the hidden widget and reorder their feed. We run test after test after test on them and they engage more with their feed when it's ranked.

Working at Facebook was full of really tough and interesting product decisions. It's the first time I've worked on a product that is so heavily instrumented, and so clearly understands the way that people using the software are using it. As a result of that insight, what you find is that how people think they use Facebook and how they actually use Facebook are greatly at odds.

[0:06:09.0] JM: What reflections do you have about building a feed? Because now today, we have feeds in Twitter, Quora, my podcast player, LinkedIn. Any broad macro reflections on how to architect a good feed?

[0:06:23.0] JG: One of the big ones is the disproportionate amount of attention goes to the top 10 slots. Rationing or using those slots wisely is one of the most important things to do. There are a lot of distributed systems problems that come with building a feed. It really matters and it will really vary whether you have a world more like Facebook's, which is relatively more peer-to-peer.

It is it is relatively the case that you have at least half your audience posting and being consumed by a finite – by a relatively small audience. Versus I think an environment like Twitter, where you have a much smaller number of people – it's much more asymmetrical. You have a smaller number of tweeters who have immense followings. Facebook has some of both of course, because of pages and celebrities. The approach you take – there's not one right architecture for a feed. It really depends on the ratio between publishers and consumers.

[0:07:20.1] JM: Facebook was based on the LAMP stack. It's a highly scaled iteration of the LAMP stack. Tell me what the downstream ramifications of that have been over the course of Facebook's history.

[0:07:33.1] JG: I've spent a lot of time thinking about this actually. I joined Facebook not from Google, which was also more or less a LAMP stack, but from VMware, which was not just writing native software, was writing operating systems. There were many things that Facebook engineers just took for granted in their release cycle that were to me, the equivalent of defying laws of physics, of a faster than light travel.

One of the biggest ones is that the cost of a release shrinks nearly to zero. I think that companies that migrated – companies like Microsoft that started life in the native software era and migrated to the web era never fully shook off the culture of developing for native. That is a culture which values an engineer, which says the definition of a good engineer is one who can call his shot and make it, one who can estimate precisely and who can deliver predictably, who says when something will be done and does it, who figures out not just what it'll take to code it up, but what it will take to integrate it to fix all the bugs who anticipates all the things that could go wrong.

The reason that's so critically important in a native software world is that the cost of a release is really high. You've got a test across a broad matrix. You've got to integrate with every other piece of – actually, you got to integrate either way. It's really the test matrix and then it's the delivery of the software, both of which come from the fact that at the end of the day, the way you deliver your software is to create this binary, to print CDs with the binary on it, or maybe find post the binary to the web somewhere where it can be downloaded, but to send those off into the world where you don't control where your end-users win and how and where they install it.

Shipping a point release, hotfix is incredibly costly not just to you, because you've got to go through that whole heavy process again, but to your customer base who now has to accept an upgrade and find time to test the upgrade themselves and on and on. Getting the release right the first time is nailing the landing, sticking the landing is incredibly important in the native world. It just doesn't matter at all in the web.

First of all, there's no test matrix because your software is only ever going to run in an environment that you control, which is completely homogeneous, whether that's a zillion VMs, whether that's containers, whether that's the cloud. Second of all, you have full control over the deployment environment so you can ship whenever you want to. Facebook did. When I arrived, Facebook shipped every day. When I departed, they shipped twice a day. You didn't even have to wait for a hotfix vehicle if you screwed something up. You could just port the fix and ship it.

When the cost of releasing goes to zero, and I will say that Facebook also had another advantage going for it, which is that it's essentially free software. Small glitches and bugs have – if Facebook's down for five minutes, it's just they lose some advertising revenue in that five minutes, but it's just not the end of the world for their users. Customers aren't going to abandon them. Users aren’t going to abandon them over a five-minute outage.

Between the two, they can accept far more risk and that means they can go much, much faster and they can have a software lifecycle that is optimized for speed, not for predictability. I meet startup CEOs all the time who asked me, “Well, how can I make sure my engineering team hits their deadlines?” I say, “Is that what you want?” “How can I make sure that my engineering team is productive?” To them, that means hitting deadlines. I actually think it's the opposite. I think that predictability comes at the expense of productivity. I think they're a literal time-space tradeoff.

What I saw at Facebook was software ship unreasonably much faster than I could have imagined at VMware, and some of it was that you didn't have a test matrix. You tried something and if it worked in your dev environment, it pretty much worked in production 99% of the time. It was also the case that not having to think about hitting that release cycle gave you so much more flexibility. You could be on the balls of your feet. You could say, “Oh, the photos team needs help from newsfeed team this week. Sure, I'm going to send an engineer to help them, because I'm not going to miss my huge release date in the sky if I have to reshuffle tasks and reprioritize.”

Just that ability to move people around very fungibly to unblock each other, tends out to be a huge turbo button for the entire engineering organization. We don't realize how much overhead we're sacrificing to people feeling adherence to deadline pressure that doesn't allow them to reprioritize on the fly and doesn't allow them to help each other out and unblock each other. I think it's actually tragic how many web stack companies, LAMP stack companies are out there who are still slavishly adhering to the deadline, which is just this legacy, these shackles that come to them from the bad old native days. They don't have that constraint, but they're still giving themselves the –

There's other reasons to have deadlines. If you are a B2B company and not a consumer company, then you may have revenue recognition issues that require you to ship features to a customer when you said that you would. By and large, the features will come faster if you have a good iterative process and not if you have a deadline-driven process.

The other nuance here is feedback loops. Everybody knows you should integrate early and often that when your code touches other people's code, you learn new things. Also, when other people see your code and experience it and use it, you learn new things and you fix it. If you go a year between releases, you will get all this feedback from the market that you can't adopt and recover from in time to get into your next release. If you go a month between releases is actually still really slow to get feedback loops.

Facebook more than any other company I've seen has embraced this idea of real-time feedback to what you're building. The way Facebook pursues designing a product is incredibly iterative. Put together our first prototype quickly, we ship it, well first of all, it's in the production codebase.

Facebook only has one branch. We use feature flags to – so everything's running in production all the time, but we use feature flags to control who sees it, which means that your code is always integrated with everybody else's code.

You're also getting immediate feedback, first perhaps just from the engineer, the development team itself that is building the feature, but then maybe you open it up to other Facebook employees, then maybe you open it up to a small – perhaps you need feedback from real users, so maybe you turn it on to 0.05% of users and see how it gets used. Or perhaps it's only useful – perhaps it's a feature that involves connections between people who are both friends. You don't want to pick a random 0.05%. You pick a small English-speaking country like New Zealand and you turn your feature on there and see how it's used and how it affects the overall metrics.

This enables Facebook to have real-time feedback on every product that they're developing as they're developing it, or did between 2010 and 2014. I can't speak for the Facebook of today. Which in turn, enabled us and we had no deadline pressure around our necks. Those put together allowed us to be incredibly agile. They allowed us to always reserve the right to wake up smarter and make a better decision tomorrow, and one that was based on real actual usage.

We didn't commit. I think where a lot of product teams get in trouble is with sunk cost. We commit all this time and energy and effort making our perfect beautiful product. Then by the time we actually get feedback on it, we're so committed. We’re psychologically committed, because we worked for months on it and we also got this immense codebase that's really hard to change. By being really minimal in our approach at Facebook, the concrete was still wet until incredibly late in the game.

It took us a year to ship the new newsfeed, but that amounted to I would say about six or seven four-week cycles where week one was a prototype, week two was testing it on Facebook people, week three was getting some form of external usage data, whether that was just pumping real production data through it, or actually showing it to production users. Week four was trying to polish and iterate on what we learned. At the end of week four, we decided to prune that branch and start over.

It took us six or seven tries to get to what we thought was the right design in that way. We spent about four weeks each evaluating six or seven designs, landed on one that we thought was best and then spent I would say three or four months really hardening it, scaling it, making it ready for production usage. Running along that same time horizon, the machine learning team was working in the background to really get the ranker built. That was much more of a six to nine-month end-to-end effort all focused on one area. Those came together at the end and we were able to ship them around the time of F8, Facebook's big conference.

We did have some deadline pressure towards the end and that was to sync up with the release of Open Graph. That combination of being LAMP native, Facebook really jettison what I would call some sacred cows for most software engineering organizations. The deadline and the idea that you get feedback at the end. I've never seen another organization that incorporates feedback along the way like Facebook. That company can turn on a dime.

A lot of it is enabled by the architecture. A lot of it is enabled by the release process. I do think some of it is enabled just by culture and personality of the early team. I think that Mark has a lot of gifts, but I think one of his superpowers is to really admit that he was wrong and not to stick to an old idea that turned out not to work.

I think as humans, we all have this blind spot. Once we've stood up and put our name on something and said X is true, it becomes really, really hard for us to ever retract that message, or contradict ourselves, or admit that we were wrong and it's really why.

[0:17:37.8] JM: I don't like the vindictiveness of the way that the Facebook story is being told right now. Innovation, people take wrong turns or take dubious turns and that's how you get progress.

[0:17:54.8] JG: Something I wrote about innovation a little – just right after I left Facebook was look, innovative ideas by definition always look wrong. They always look crazy. Because if it looked like a good idea, it would be an obvious idea, right? By definition, it would not be innovative.

What makes it innovative is either people think it's not possible, or they think it's a bad idea, or they think it's – it would be bad for the world, or they think that it can't work, that it's too risky, that it would have too many – basically, either it's impossible or it has too many bad side effects to be worth trying. Otherwise, someone would have done it. Otherwise, it would be like, “Well yes. Of course, we would do that.”

It's really hard to tell the two apart at the beginning. I don't think there's anybody who can reliably distinguish good innovative ideas from bad ideas. I think you just have to try them and see what works. Within maybe a values framework of some things we just don't try. I think that to innovate, you must be willing to embrace the risk of being wrong and you must be willing to embrace the risk of failing. Otherwise, innovation doesn't happen.

I do think that Facebook's been an incredibly innovative company, in part because its architecture, its process and its engineering value system enabled it to take risk and not just embrace risk, but encouraged everybody to take a lot of risk. I do think that that was one of Mark's great gifts as an entrepreneur. It's one of the things that I think makes him one of the great entrepreneurs of all time.

When you take a lot of risks, you can get a lot of things wrong. Facebook's genius is not that it's right any more than anybody else. It's that it tries experiments at a higher velocity than anybody else and they purge the ones that don't work faster than anybody else. It's no surprise really that in the end, the deep-seated challenges for the company have been problems that don't show up immediately, problems where the downsides were credibly lagging indicators, or finding out in 2018 about a problem that happened in 2011.

Facebook's phenomenal at optimizing, at making curves go up into the right, at improving on anything you can measure. I think that does give them a little bit of a blind spot about evaluating and optimizing for things that cannot be measured, like trust, or data privacy, which is not to say that the company doesn't value those things. I think they do incredibly deeply. It's just that most of the tools in their toolkit don't solve those problems. They've got to build some new tools now.

[0:20:28.5] JM: I want to come back to what you said about Facebook's ability to turn on a dime, because I think this was exemplified when you were leading the mobile team, or the transition to mobile team I should say. For all of Facebook's ability to experiment and explore lateral movements, this was a time where Facebook really had a necessity put against it. You need to move to mobile. It was fortunate that the leadership inside the company saw that this was important and you were heavily responsible for leading that effort. Tell me about the strategic inflection point of moving Facebook on to mobile.

[0:21:08.3] JG: I would say the years I was at Facebook from 2010 to 2014 were the transition to mobile years. While Facebook was good at turning on a dime for product changes, or even strategic decisions, the mobile turn was more out turning on a truckload of dimes. think it took us all of those four years. I would say in 2010 when I first got there, the resources allocated to mobile were just meager.

Everybody could see that mobile was rising, but it was still a small percentage of the whole. It might help here to explain a little bit of Facebook's organization, because that I think shows a little bit of how decisions were getting made. There was a big infrastructure organization of people building back-end services and operating the data centers. There was a big product organization that was divided into many pretty autonomous units. Newsfeed was its own autonomous unit. Search was its own autonomous unit. Messaging was its own autonomous unit. Groups had just been created and was its own autonomous unit.

There was a lot of latitude for product teams to really own their own metrics and own their own endpoints and do whatever they wanted and achieve great results. That was one of the things that made metrics go up into the write really fast was lack of interdependencies between groups. There weren't lots of layers of authority and there weren't lots of people who would say to the newsfeed team, “No, you can't do that.”

Mobile was one little product team and it didn't have a whole lot of resources. Basically, they spent all their time looking at what was being built by these product teams for the web and desperately trying to port it all over to mobile as fast as they could. At least that was the story on iOS and I won’t tell you about Android, it's too embarrassing.

Somewhere between 2010 and 2011, it became obvious that mobile was really important. I picked up the photos team in 2011 and it was really clear in 2011 that mobile was becoming a very important platform for photos, because all of a sudden, you weren't getting big album uploads from – people take their digital cameras on trips, take a ton of photos and then you'd get this huge album uploaded with hundreds of photos from that trip, right? That used to be how photos got on Facebook.

Now you're wandering around and the camera device is the device that's running Facebook, and so you're getting a lot more one-off photo plus. I take a picture, I post it. We started to see that emerge. The photos team knew that mobile was an important platform, maybe before it was generally obvious to other product teams as a whole. It was becoming more and more clear. I think towards the end of 2011 we're like, “Okay, we got to do something about mobile.” Well, we want to take mobile seriously, but we've got all these engineers, we've got all this investment in PHP, we want to go really fast, we don't want to have three duplicate code bases, so we're just going to bet big time on HTML5.

What we did was we chose what would work really well for Facebook, what we knew how to do. We knew that the LAMP stack gave us all these superpowers that we just talked about, the ability to go fast, the ability to iterate quickly in real-time. That was our first try. It was the right decision for Facebook's development productivity and release process, but it turned out that despite all that in 2011, HTML5 could not meet what people using Facebook wanted, which was the performance and snappiness of the interface, as well as some key features like integration with the camera and SMS and other sensors that you couldn't get from HTML.

We made the very difficult decision in 2012 that we would scratch that. That was probably, I don't know, a six to nine-month wrong turn. Our first pivot to mobile was the HTML5. Second pivot to mobile was acknowledging okay, we've got to go native. By then, we had finally hired a few more people. I should say, the HTML5 decision was also really rational when you considered that we had something 3 or 400 engineers that could do PHP and five engineers that could do iOS and two that can do Android.

Okay, so by 2012 we've hired more people who know native and we're going on a hiring spree and an aqua hiring spree to get more. Now we're closer to something like 20 people who know native. We take them all and we put them in a war room and we say, “You're going to go make amazing native Facebook apps. Go." They did. They sprinted. They were heroic and they built really good Apple and Android native applications.

It should be said, most of the people working on those apps were new to Facebook. We had hired them from companies like Mozilla, from companies like Apple where they had experience developing native. The core Facebook product engineers who know PHP first and foremost and JavaScript, that crew is largely sitting outside the war room, working on the web, but watching as if you identify as part of the newsfeed team, you're not identifying as part of the web team. You define your success by how many clicks, likes and comments are driven by the feed today. What are my daily usage of newsfeed?

Or in the photos team, you sitting there trying to maximize metrics like participation rate. How many people upload a photo? How many photos get uploaded? You're sitting there in 2012 watching mobile be a larger and larger portion of the metrics that you live and die on. You have to go knock on the war room door and ask nicely to get a feature on those platforms.

There was also a lot of demand to train up more mobile engineers, so they wouldn't be as much of a bottleneck, but the product teams all have the appetite now to contribute to the mobile apps the way they had not in 2010. We have these mobile native apps that are shipping. I think probably yeah, it was maybe late 2011, early 2012 and they ship. We open up the war room. We create a mobile engineering team. We start getting more and more. We try to train every product engineer at Facebook and doing mobile. We create these two-week classes and we of course, continue trying to hire every mobile engineer we can get our hands on. We made massive alterations to how we recruit and interview, all with the intention of being mobile first. That I would say is the second big pivot to mobile. It worked and it produced good apps.

Where it did not work was okay, now what? We're out of the war room, these product teams desperately want to control their destiny. It is totally visible from space to them now that their future is mobile, because they can't hit their product specific metrics, because mobile is just this rising tide that is driving every metric that matters. They are running into the mobile team as a gatekeeper.

Now remember how I said the Facebook way of delivering software was enabled by the LAMP architecture and by the Facebook process and by this idea that the cost of a release is zero? I think that a lot of Facebook engineers grew up with that. They'd only ever worked at Facebook, or maybe they worked at Google for a year or two and then were at Facebook. There's a lot of Facebook exceptionalism. We do it this way, because we're Facebook, because we're bold, because we embrace risk.

I think there was not a lot of awareness oh, we can do this because we're a LAMP stack and not a native stack. I don't think that was a widely understood feeling amongst those product teams. Meanwhile, these mobile engineers had been through Facebook boot camp, but they had gone – they'd been tossed straight into the mobile war room. They'd never worked on anything at Facebook but the native app. They came from places like Apple, that has – if you can imagine a diametrically opposite engineering culture and software delivery lifecycle, it's Apple, right? They're still in the waterfall world. They have operating system DNA. They have hardware DNA.

[0:28:59.5] JM: Proudly in the waterfall world.

[0:29:00.6] JG: Yes. Even coming from something like Mozilla, or any other – they came from a native development background, and so they came with a value system that was you call your shot, you design it and you perfect the design and then you build it and then you test it and then you have things like UI freeze and feature freeze and know – and then you have a release branch where all you're doing is closing bugs. No, you can't port your feature changing to the release branch just because you got your AB test results came back and told you that the feature should be different. Of course, you can't do that.

They're running these eight-week release cycles. The product teams are desperately trying to contribute code and develop features for these native apps the way they had always done it for web. Because we've taken a PHP engineer, but more importantly a Facebook product engineer who's grown up with the Facebook release lifecycle and teaching that engineer objective-C and iOS libraries, or Java and Android frameworks doesn't actually unteach all the release process change.

We've said, “Hey, we have this release process. Now you have to jump through all these hoops,” that really just felt like bureaucracy to those Facebook guys, right? Magically we’ll get good native apps out of this. When I arrived at the helm of the mobile engineering team, these two groups were at daggers drawn. You had the mobile engineering team that identified themselves as the gatekeepers against the barbarian hordes. Yes, the Facebook product engineers were the barbarian hordes who are making work for them, who are really sloppy and slapdash and who are creating lots of bugs.

Another thing that's different about a native app versus an endpoint is I mentioned that newsfeed could have a lot of autonomy, because we own our own endpoint. If we screw it up, we've just hurt ourselves, right? Or the chat people can take all kinds of risk with chat. They have their own news feed if they mess up chat on the web. In native, we're all one payload. Another way that risk can pound is well, if you introduce a crashing bug, that crashes the app for everybody.

There were so many reasons why you had to be far more risk-averse, that were fundamentally technological not cultural. It was seen as a cultural schism. Both sides are sitting there viewing the other side as having bad values, bad tastes and posing an existential threat to the company even, right? You people want to ship horrible software and you people want to be bureaucrats and stop us from iterating.

Remember, I told you the story about the newsfeed development process and how it was six or seven – to evaluate any given design idea, you evaluated it in these one-week cycles. Those one-week cycles were driven of course by the main branch getting shipped every Tuesday. You would do four of these one-week cycles to fully explore an idea and then you might do four or five of those ideas to land on the right idea. Now imagine that your fundamental cycle time unit to get feedback on an idea to be able to ship an AB test for example is not one week, but eight weeks, because that's how long the mobile app – I mean, we were pipelining them, so that a new release went to the App Store every four weeks. That just means if I ship something on day zero, by the time I get some feedback, I'm already after the feature freeze cutoff. I'm already on the release branch for the release that's four weeks away. I'm supposed to make changes on main that are going to ship eight weeks away.

To take the fundamental unit of product iteration from one week to eight weeks blows up my one-year project to an eight-year project. I mean, it's just unthinkable. The product teams are sitting here looking at the mobile teams and they're just baffled. It's like, “No, you can't do software that way. What are you talking about?” The other guys are, “No, you can't do software that way. What are you talking about?”

Somewhat surprisingly to me, I looked around and realized that I was one of the only engineering directors at Facebook that actually knew what both sides meant. They were just talking past each other, because they came from these very different backgrounds. If you looked at the Facebook engineering director ranks, there was the guy who's leading the mobile charge was someone who had come to Facebook and had been from native software and had – there were two. One had come from Mozilla, one had come from a startup, but they both had only done native at Facebook.

If you went and looked on the product side, almost everybody had a background like if they'd worked out somewhere other than Facebook, it was Google, or was another LAMP stack company. There were one or two that had – a year or two at Microsoft long, long ago when they were engineers and not managers. I was the only one who had run software releases both for a native software company VMware and for Facebook product team. I think me and maybe Trep could say that. Maybe Jay Parikh, the head of the infrastructure org. Sitting in infrastructure, he was ill-position – actually, he had never done a product release in Facebook.

I was somewhat uniquely in position, although I had not worked with mobile tech before to understand native. This was really a process issue as much or more than it was a technology issue. It was also a culture and an organizational design issue. I stepped in to helm the mobile engineering team and it was the – it became immediately clear to me when I went to talk to my peers from the product organization how frustrated and stymied they felt working with the mobile organization. It was immediately clear talking to my new teammates in mobile what a bunker mentality they had, like how much they felt under siege from all the product teams trying to force changes in to the mobile apps.

Well, the mobile team was held responsible for the quality of the mobile app. The product teams were only held responsible for their product metrics. The mobile team was held responsible for things like the App Store reading off the app, or the crashes and performance of the app. It was a terrible misalignment of goals along with everything else. My stint at the helm of mobile, I consider the third and final pivot to mobile. I said when I took this job okay, we've been talking for two years about being mobile first. I took the job in early 2012.

We had been talking for two years about being mobile first. Guess what? A mobile first company does not have a mobile team. Facebook never had a web team. We had a newsfeed team, we had a messenger. Web was such a baked in assumption that every product team did web. If we are truly mobile first and every product, then every product team is a mobile team.

[0:35:31.5] JM: Yeah, you can have a framework team, a platform team that supports the product teams. We had that on the web. It's not a mobile team, it's just like an infrastructure, a product infrastructure team. I took the job knowing that my goal if I was successful would be to marry Facebook release process and mobile real and native release process. In other words, to evolve a new and third process that was not trying to take Apple process and somehow make it work for Facebook, because Facebook really is different than Apple. We're not shipping hardware. We are going to have another release coming in four weeks, or actually on my watch we made it two weeks, which really eased a lot of pressures too.

It is also the case that Facebook can't just ship native software the way they should LAMP stack. You cannot ship twice a day. You cannot let the messenger team crash the app and blow up newsfeed. We had to evolve something out of the merger of these two sets of ideas. We also needed a heck of a lot more automation and tooling to make it less painful. If we had to have a QA cycle on native, which Facebook on the website they didn't have QA. There was no QA organization. There was no such thing as a QA cycle. There was no QA gate keeper or decision maker. The developers were responsible for fixing bugs and discovering bugs, just like writing code. You discovered bugs by either your co-workers reporting them to you, or watching the metrics. If a metric plummeted, you realize you had a bug.

[0:36:59.6] JM: Which is a stark difference than the avalanche of unit tests and integration tests that were often taught in other places.

[0:37:06.9] JG: Well, Facebook did have a lot of testing automation. I mean, developers were responsible for doing that. There was a strong testing religion. I think that you get that when – but unit testing religion, not manual testing. That's what you get when you ask developers to own test, instead of QA.

I don't want to say that Facebook wanted to ship a lot of bugs. They had a lot of mechanisms and actually really great and wonderful tooling on the web side to find and prevent bugs, but the idea of a test matrix of needing to exercise the same functionality on many different platforms on the three versions of iOS and the five versions of phone and the God help us, infinite flavors of Android and Android devices, that was completely new to Facebook. That was a thing that had to be accommodated. The ideas of feature freeze and letting the codebase take only bug fixes for a little while before. That was needed to be accommodated.

The idea that that you'd have to go to a release team for approval to cross-port code, as opposed to just moving it into the branch. All those things needed to be accommodated for something like native, where a release is not happening every day. It's happening every two weeks, maybe every four weeks. Then the other big change that had to be made was mobile could not be the only one responsible for quality.

In fact, we already had and this just speaks to the fact that the mobile team was new to Facebook. They didn't understand what tools Facebook already had that were authentically Facebooky to solve these kinds of problems. On the web side, all products teams signed up to performance goals for how fast their pages would load. They signed up to reliability goals for how frequently their pages would crash, or would not crash. They signed up to code quality goals and efficiency goals on how heavy your feature weight was on the servers. How many new servers do we have to buy it around your feature? They were doing that. You could distribute responsibility for quality. We have a lot of mechanisms and buy-in for doing things like that.

The mobile team just had this bunker mentality that the product teams didn't care about quality and would not accept – didn't accept responsibility for mobile quality, but they hadn't really tried to get them to accept responsibility for quality. I walked around all the product teams and said, “Okay, you know how you have a performance and reliability goal for web. Well, I'm giving you them for the apps." They're like, "Really? Oh, well yeah, I guess that does make sense." I mean, there was not any pushback. It was just they've not been asked.

I would like to say there was a single silver bullet that enabled the pivot to mobile, but it was more a hail of lead bullets. It was a lot of lead bullets. It was getting enough engineers who were trained that each product team had competent mobile engineers on their bench and not incompetent, half-trained engineers. It was getting enough tooling and frameworks, so that we could automate a lot of the testing.

It was getting the release cycle time down from eight weeks to four weeks. Actually, one thing that really helped was transfers, like transferring people from product teams into the mobile engineering team to learn the right ways to do things. Transferring people from the mobile teams into the product teams really reduced the us and them mentality, because it's like, "Do you trust the messenger team?" "No, I don't trust them at all. They're barbarians." "Well, do you trust Ben?" "Well, actually I trust Ben a lot. He used to sit next to me." "Okay, well Ben sits in the messenger team now. When you have a problem with the messenger team, instead of hunkering down and saying how awful they are or trying to escalate to a third party, why don't you go talk to Ben?" "Oh, I'll do that."

None of these changes is going to sound – another thing we instituted was a countdown process. One of the goals that felt really difficult to achieve was how do we improve the App Store ratings of these applications, because they were low? There was a lot of voodoo. Everybody had their favorite pet bug that they thought was driving low ratings. We actually borrowed a data scientist and scraped all the text of all the reviews, because Apple doesn't give you an API to get the review text. It's terrible. We had to scrape it and then just run some NLP clustering on it and actually, find a quantifiable way to prioritize what are the top 10 issues driving low ratings.

Well, lo and behold, the first two are just what you think. The app is slow and the app is crashing. Those are actually really good ones, because it's really easy – not easy, but we had a big effort to measure performance and to measure crashes and those could be attributed in a sign to product team, because you know the page that was loading slowly was the newsfeed team, or we can instrument the warm start and cold start time and figure out whose code is contributing to it, so we could actually hand out performance goals to every product team and we could hand out oh, frequency of crashes. We're getting all these crash logs. We can assign those to product teams to fix.

Yeah, there's going to be some scary, ambiguous ones and those the mobile engineering team will keep. Then once you would never imagine, the number four in 2012, maybe 2013, the fourth largest complaint about the Facebook application was Candy Crush won't connect to Facebook.

[0:42:13.6] JM: Oh, no.

[0:42:15.3] JG: The reaction from everyone was like, “Well, that's not our bug.” I'm like, “Yes, but getting a four-star app is our goal. This is the fourth largest bucket of complaint, so let’s go fix this.” Lo and behold, we have a platform team who works with third-party developers. Lo and behold, they say it's a Zynga bug, not a Facebook bug. Fine. My guess is we have enough leverage with Zynga that we can call them and get them to fix their bugs and we can collaborate on this. Lo and behold, we start getting in motion. Oh, actually there are some Zynga bugs and there's a Facebook bug or two.

[0:42:48.7] JM: Of course.

[0:42:50.4] JG: Fine. Boom, we can make a huge cluster of pain for our users just go away.

[0:42:56.4] JM: Wonderful.

[0:42:58.9] JG: It was nothing the mobile engineering team could ever have done a darn thing about. Because it's all back-end server code owned by the platform team. Android photos not loading was another huge one. Just being able to shed insight onto what caused user pain actually enabled us to build these great cross-team collaborations to go solve those problems.

While these teams have a lot of differences and culture clash about what it means to be a good engineer, or a good citizen of Facebook, one thing a 100% of us agreed on, the real common ground was we wanted to make these applications great for the people who use them, for the Facebook community. That drove and motivated everyone.

These folks who said, “Oh, I can never collaborate.” It was trying to get bipartisan legislation through, right? It's the one thing Democrats and Republicans can agree on is job creation. The one thing mobile and product engineers could agree on was let's do right by the Facebook community. That really was the bridge. The end of my tenure running the mobile engineering team was I declared success and I reorg the mobile engineering team out of existence.

We sent bunches of mobile engineering teams to all the different product teams and we kept a core of it and said, “Great, you're now a mobile infrastructure team. Your job is to create frameworks in performance tooling and quality tooling and to operate this release process.” We're not going to own features or functionality. Features and functionality are going to be owned within a team that lives and dies by the metrics of those features. That is integrated end-to-end that owns the front end and the back end and the web front end. That was the third and final and successful pivot to mobile. Only took four years.

[0:44:49.8] JM: Now with the mobile story you just told, there were a bunch of byproduct effects from fundamental strategic pivot that led to I think some cultural changes, some engineering org changes in the company. You also were responsible for helping with changes to boot camp, changes to recruiting. Was there a point at which you changed to macro level org changes, or were you always working on a product team and then simultaneously helping with org changes, like stuff to recruiting and stuff to boot camp?

[0:45:19.7] JG: Recruiting and boot camp were things I did throughout my four years at Facebook. I mean, I think all Facebook managers – I don't know. I just viewed it as a communitarian, I should be running my org, but I should also be doing things that benefit the entire organization. I think that's one reason why the CTO viewed me as a key lieutenant.

There was a period of about 10 months when I stepped away from. This was in 2011-2012 when I stepped away from running a product team and I focused exclusively on hiring, onboarding. Actually, this was another turnaround. Actually, my whole tenure at Facebook was turnarounds. I think some people at Facebook would actually say this one was even more important than mobile. I took over the tech recruiting team for a year. We had them report into engineering for a year. I gave them back when I was done. They'd never met a recruiting goal.

We had gargantuan engineering hiring goals and the goals kept doubling because that was the rate we were growing at.

I think the year and I can't remember what the miss was at the end of 2010. At the end of 2011, they were supposed to have hired 300 engineers and they hired 250. Then we had told them the goal for 2012 was going to be 600 engineers, twice as many as the 300 that they had missed. They were trying to ramp up and hire more recruiters and sweating really hard. Then we decided to IPO in 2012. We both felt that that would give us more cash to hire more engineers, but we also were worried that people who were finally liquid that there could be a lot of attrition. The people could just make so much money, they could just retire and that we couldn't – that we had to increase our assumptions about how many people would need to be replaced.

We came back to them in December and said, "Oh, actually your hiring target for 2012, remember we told you it was 600? It's really a 1,000." Recruiting's reaction was like, the prisoner on death row being asked what they wanted for their last meal. I mean, they just felt doomed. There was no way for them to succeed, because December is already too late to hire and ramp a bunch of new recruiters. By the time you've got the recruiters hired and ramped, you're halfway through the year and you've missed the H1B window, because you've got to apply for the Visas by April and you've missed the new grad window.

They just felt really doomed. It should also be said that Facebook's interview process had not significantly evolved or scaled in years. I think of these kinds of organizational systems and processes, they're not so different from architecture. If you have a pile of code and a set of systems and an infrastructure that is scaling to you're a 100,000 visitors a day. At some point, you've got to get to a million visitors a day. For a while, maybe going to 200 or 400,000 users, you can just throw more hardware or more cloud instances at it, right?

At some point on your way to a million users a day, instead of a 100,000 users a day, you've got to stop and rearchitect. You've got to stop and fundamentally rethink where are the bottlenecks in my system? I've got to move them around. Because you just cannot keep throwing more hardware at the problem. It becomes outrageously expensive and in some cases, your software just can't horizontally scale. It hits limits. It hits the wall.

I would say that we had an interviewing and hiring set of processes that were designed for a much smaller hiring target and a much smaller company, a smaller group of potential interviewers, a smaller group of recruiters, people who knew each other personally and directly. We had hit and we kept trying to scale that horizontally by hiring more recruiters and we had hit a scale wall, where we needed to rearchitect how we were even thinking about interviewing.

One of the things that was really obvious about our interview process just symptomatically was that we had tons and tons of false negatives. What I mean by that is that we were rejecting lots and lots of engineers who would have – who should have been hired, who were over our hiring bar. The most visible symptom of that was employee referrals. You'd refer someone to work at Facebook who you had worked with before and you knew was good, and that person wouldn't make it through the phone screen, let alone get a job offer.

There were really widespread complaint. Maybe sure, fine, one person just like, it's your buddy. You think he's good. He's not that good. The complaints were really widespread enough that it was obvious we had a problem. It was pretty obvious that that problem actually was largely happening at the phone screen stage. It makes sense. At the on-site stage, there's four interviewers, there's multiple checks and balances. If one person is too harsh, that's okay, there's three other people saying this person really was good.

Usually what happens in that situation where it's a split decision is either you decide you don't trust the interviewer voting no, or maybe you send them a programming test to take home, or you bring them back for some tiebreaker vote, okay? In a phone screen, there's only one interviewer and if they're too harsh, it's a single point of failure.

The other problem with phone screens is they're much lower fidelity than an in-person interview, so they're much more prone to wrong answers. Yet, another problem with phone screens is that nobody good wanted to do that. Because at the phone screen stage, you're seeing a much wider variety of candidates and a lot of them are not very good candidates. The people who were senior and good interviewers said, “Well, I want to reserve my time for the important interviews,” that is the on-site interviews. Who was doing the phone screens was people who were junior and new to interviewing, and who were much more afraid of bringing through someone bad to the on-site interviewers and looking bad in front of them, than they were afraid of missing a hiring target that was really abstract to them. Because as far as they're concerned, their team grows when new people come out of boot camp, right? They felt really attenuated.

The interviewers were so attenuated from the hiring process that they didn't have any motivation to get people hired. The overwhelming and of course, there's the MO that all good companies have, which is you should have a high bar, you should be really selective. The MO was for a phone screener when they were uncertain, they would vote no. That meant that we were voting no all the time on phone screens that should have been yes. I ran some experiments that proved it. I could go on and on about recruiting.

[0:51:32.2] JM: I'm sure you could.

[0:51:32.8] JG: We got a lot more effective over the course of that year.

[0:51:36.0] JM: What do you miss most about working at Facebook?

[0:51:38.4] JG: Oh, the people. The people were phenomenal.

[0:51:40.5] JM: Is there any difference between people and culture?

[0:51:42.3] JG: Yeah, yeah. I would distinguish those. Yeah.

[0:51:45.6] JM: What do you mean specifically about the people? Like individuals, just friends that you belt, or –

[0:51:50.2] JG: Sure. I think that the collective caliber of the people I got to work with was so high. You know how I said, you can just – when you know the price of a release is really cheap, you can just make so many assumptions, you can just live life a different way. You can be fundamentally more productive and higher velocity and make decisions faster and take more risks.

When you can count on the fact that every coworker is competent and collaborative, you can go so fast. You can have so much trust. You can feel so safe. I show up at Facebook when it was probably 500 or so people, maybe 600 across engineering and product and design. When I left, I don't know, maybe 3,000 across those functions that were my primary co-workers. At those sizes, you can't guarantee that for a 100% of people. Man, it was 90 plus percent.

I don't know how to describe how few politics Facebook had in 2010. Even across 600 people, there should be some selfishness, there should be some agenda, there should be some people saying I want to recruit for my team.

[0:52:59.7] JM: Why is that? I mean, this is consistent with the other interviews I've done that there was not a Game of Thrones mentality. There were people who were able to prove that they were good engineers and those people naturally rose to becoming leaders and those leaders were able to lead other engineers to build products efficiently. It seemed quite a friendly, non-competitive atmosphere.

[0:53:24.4] JG: I'll give you two classic examples where all managers – whether they're like – they don't have to be political ourselves. They just have to be well, I think it's my turn for a new hire, right? Facebook had this system where engineers not recruited to a particular team. They're recruited into boot camp. Everybody goes into boot camp for the first [inaudible 0:53:41.7]. Yeah.

Not exactly though, because there's not an external sorter. You pick which team you join and actually, that was true, right? Managers would come and recruit out of boot camp. They didn't recruit from the external world. They would be part of the interviewer pool like everybody else, but new candidates including your former interns would go into boot camp. Then you'd go around week four and start talking to different boot camp. If you had openings in your team, you'd go talk to boot campers and pitch your opening and the boot campers would pick.

We had a way of putting our thumb on the scales, which was they were always in any given week, and the number varied with the number of boot campers. Say there would be four teams that Facebook would communicate to the boot campers, these are high-pri teams. If you go to one of these four teams, it's most in need, it's crucial to Facebook success, it's a pivotal moment for that team, these are the teams that we really want you to think hardest about, because they are highest priority to the company.

The pitch to boot campers was pick the team that's at the intersection of where you're passionate and where you can have an impact. There were also boot camp mentors who worked with each. During the period of boot camp, you don't have a boss, but you have what's called a boot camp mentor, which is an engineer who's shepherding you and helping you not just get the lay of the land, but get your build environment up and get tasks assigned to you and do work. Then later on, they can be a little bit of a local guide about teams.

The boot camp mentors were always pulling for the high-pri teams, which mind you, may not have been the mentors teams. There was this very collegial – I remember picking up the search team and saying, “Oh, should we go try to get some boot campers?” The manager, their search team was like, “Oh, we're not on high-pri, so we should steer boot campers who come to us to go talk to the high-pri teams.” I'm like, “Where does that come from?”

There really was this spirit on the engineering side of trying to globally optimize, rather than locally. I think it came from a few things. I think it came from a flat organization. When everybody is within line of sight of the head of engineering, it's easy to think about having the same value system as the person running engineering overall. That can't scale, but it did for a long time. I think it comes actually from the boot camp hiring system itself. When you accept a job offer with Facebook, you're not accepting a job offer with the search team, or the messenger team, you're accepting a job offer with Facebook.

People became citizens of Facebook first and their team second, because they made that decision a month later when they were graduating boot camp. That was a really big difference from VMware and trilogy and the other places I'd worked, where when you joined VMware, you were hired by a certain team. You very quickly as VMware grew, not so much in the early days, but as VMware grew larger people developed a partisanship for their own team, patriotism. Not anything like super political, or sharp elbowed, but it was just – it's like people rooting for their own sports team, right?

Well, I'm in the Bay Area so I'm going to be a Warrior's person. Oh, you're from Ohio, I guess you like the Cavs. Okay, we can be friends, but I’m rooting for the Warriors and you’re rooting for the Cavs. Yeah, we're both US citizens. It was surprising to me how much being a citizen of

Facebook first and the search team second I think changed those priorities in a really virtuous way. I don't think that has been able to – I don't think that's still there, but I think it was there in 2010. Because largely it was old-timers in leadership positions, because of the impetus to have homegrown management, I think even in 2013, 2014 almost everybody in a leadership position had joined Facebook in that way and with that mentality. I think that really kept it going for – that's how we defined gravity for so long.

I guess, the nature of boot camp, the nature of Facebook citizenship first, I think home growing the leadership so that the leadership all role modeled that mentality and incentive system. I think another factor was that Facebook made transfers relatively easy. That both relied on people to have a global optimizing perspective, because transfers are one of those things that's like, it's obviously great for the aggregate, but terrible for the team that gives up on engineering, right?

Transfer engineering from here to there is usually net positive for the organization, because the engineers leaving for a reason and they're going to be happier. Then the team that improves is at least as much better off as the losing team is worse off. In principle, it should be really great and healthy for engineering organizations to have low barriers to transferring. In practice, the source manager who's giving up an engineer is losing a lot, especially in a world where – I'll take it back to the livestock. Especially in a world where predictability and deadlines are really important. If I lose an engineer right now, I'm going to miss my deadline. Dude, I can't lose that engineer, so I start being selfish.

In a world where I don't have deadlines where I'm sort of where it's really fluid, what I'm delivering and when, I'm just going as fast as my team enables me to go. Then if I lose an engineer, it's not like my – "Okay, now I'm doomed," right? "Now I've missed my goals." It's like, "Well, my goals are going to be fluid," and then I'll go try to get somebody. Also, I'll go get somebody out of boot camp, so maybe it takes me two weeks to replace you instead of six months, because I don't have to start with sourcing a new candidate from the outside world."

There was a program called hack a month that I think made transfers easier, because it made it a two-step process, so the manager losing the engineer didn't feel right away like it was a loss. There were a bunch of small things that made transfers easier. Then I think transfers made people better global citizens, because now you have this social graph that overlays the org chart, right? You have very close ties to people in other teams. You have a lot of trust for them, it breaks down silos. Because those are your co-workers that you work side-by-side with in the last release and now they're over in the other team.

This was a principle I'd observed in Facebook long before mobile was an issue, but I just observed that the transfers meant the social graph was much more dense than the org chart. We had so many more strong ties than the org chart itself from those transfers and from boot camp. That's why it was one of the tools that I had available to me that I knew I could use later on. In fact, it was the mobile situation that was perverse where you had this one island in the social graph of employees that were new and had only ever had affinity with mobile. I think again, it was a number of factors had to conspire to make Facebook special in this way, but they really did.

[1:00:02.8] JM: Last question, you now work as a venture partner at Zetta Venture Partners. How does your time at Facebook inform your investing?

[1:00:11.1] JG: Not as directly as you might imagine, because at Zetta we only invest in B2B startups, ones with business to business as a business model. Although we do also invest only in AI-first startups. My Facebook experiences definitely are tremendously helpful. I think they've made me – I don't have a PhD in artificial intelligence, but the experiences I had at Facebook and the ways I saw us use the technology have made me really converse up with it. I think that's quite helpful. I think there's almost every VC is trying to invest in AI and a tiny handful of them understand the technology the way I do.

Apart from that, I think Facebook really taught me to value iteration and this idea of incrementally developing an idea. I think that that is incredibly powerful and important for early-stage startups. I invest also at the seed stage, so pre-product market fit. Really having the discipline to strip back your MVP and all these terms, agile, MVP, lean, they're all out there in the water for Bay Area startups.

Just like Facebook, because it never had the baggage, or the legacy, or the early leaders were not tainted with the memory of waterfall, right? They weren't converts from waterfall. They were born LAMP. I think that as hard as founders try to imbibe this idea of MVP, a lot of time your instinct is still, “Oh, no. I've got to perfect this. I've got to get this more right. I've got to do more work and build more before I show it to customers.” I think that bias, not just a bias to action, but the bias to get feedback sooner and to course-correct and reshape and pivot based on real-world feedback was a high art at Facebook. It was the default.

If two people disagreed about something, what happened next was not a debate about who was right, it was well, let's run a test and see. The power of external feedback, I don't know, it's really easy to say and give lip service to, but there's something fundamental and ineffable about that. Being able to help startups do that I think is a gift.

[1:02:19.2] JM: Jocelyn, thanks for coming on Software Engineering Daily. It’s been great talking to you.

[1:02:22.1] JG: I’ve really enjoyed it. Thanks a lot.

[END]