EPISODE 938

[INTRODUCTION]

[00:00:00] JM: Facebook engineering is unique. Software is built at Facebook in a way that is distinctly different than any other company. In our series of shows about Facebook engineering, we've mostly covered the positive side of Facebook practices. In today's show, we also explore a bit of the downsides.

Facebook moves fast. Engineers within the company must move fast, and this can reduce the time spent on formal processes and documentation. A fast pace can lead to a chaotic environment. In this kind of environment, not every employee thrives, and product development can sometimes suffer, but there are clear benefits to moving fast, and many software organizations could benefit from moving faster.

Pete Hunt was an engineer with Facebook who worked on Facebook video, Instagram and React, and Nick Schrock worked on core infrastructure at Facebook. He was also the co-creator of GraphQL. Pete and Nick helped create the Facebook culture and they're aware of the best and worst aspects of it. They join the show to reflect on their time at Facebook and the downsides of moving fast as well as the upsides.

If you're building a software project, post it on Find Collabs. Find Collabs is the company I'm working on. It's a place to find collaborators for your software projects. We integrate with GitHub and make it easy for you to collaborate with others on your open source projects and find people to work with who have shared interests so that you can actually build software with other people rather than building your software by yourself.

Find Collabs is not only for open source software. It's also a great place to collaborate with other people on low code or no code projects, or find a side project if you’re a product manager or somebody who doesn't like to write code. Check it out at findcollabs.com.

[SPONSOR MESSAGE]

[00:02:09] JM: Looking for a job is painful, and if you are in software and you have the skillset needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that's a good fit for you.

Vettery is an online hiring marketplace to connect highly qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies access is exclusive and you can apply to find a job through Vetter by going to vetter.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you.

No more of those recruiters sending you blind messages that say they are looking for a Java rockstar with 35 years of experience who's willing to relocate to Antarctica. We all know that there is a better way to find a job. So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery.

Vettery is changing the way people get hired and the way that people hire. So check outvettery.com/sedaily and get a $300 at bonus if you accept a job through Vettery. That's V-E-T-T-E-R-Y.com/sedaily.

Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:03:59] JM: Pete Hunt and Nick Schrock, guys, welcome back to Software Engineering Daily.

[00:04:04] NS: Thanks. Thanks for having us.

[00:04:04] JM: Yes. We've been doing this series is about Facebook engineering, and much of the series is focused on the benefits of the Facebook engineering culture. I'd like to have this

conversation highlight some of the downsides, or the risks, or the costs of the Facebook engineering culture. What are the downsides of moving as fast as Facebook did in the years when you guys were there?

[00:04:35] PH: That's a really good question. The first one that comes to mind is that when an engineer is dropped into an environment that moves that quickly, it's actually pretty overwhelming in terms of like being able to onboard that quickly. I remember the documentation being pretty sparse because the documentation would rapidly fall out of date. So a lot of times, engineers would have to splunk through the git blame of the fabricator history or the comments on code review in order to understand like what was going on and what decisions were being made.

For a lot of engineers, that's not something that they're used to doing. They're used to going and reading the design document associated with whatever system they're working on. What I found when I was there anyway was there wasn't a lot of that stuff. It required a lot of work. It was something that wasn't common for engineers to come in equipped to do.

[00:05:24] NS: Yeah. I think the difference or the downside is actually on a different dimension is actually more subtle and a little counterintuitive. So far that I think someone coming at this for first blush would be there is an obvious tradeoff between speed and quality. I actually don't think that's true, because by being able to move quickly and having fast feedback loops you can actually correct mistakes quickly, and actually that fast induration cycle can actually lead to higher quality.

However, I think the true downside is short termism sometimes, where you're making decisions where velocity and speed of iteration are your primary input into the decision-making, and that can get you to sort of a local maxima if you will, where you've made a strategic error by going at a certain direction. I'm sure we'll talk about some of those.

I think the most notable one was to shift to HTML 5 for mobile. But because you're focused more on short to medium-term iteration speed or velocity, you can kind of make higher-level decisions and move quickly to a point where you've done enormous damage to yourself and

kind of gotten into a – Like boxed yourself kind of into a pretty bad spot and move there very quickly.

[00:06:44] JM: We've covered the HTML 5 story in some detail could. Could you both give another anecdote that perhaps exemplifies a time when you remember at Facebook where maybe things – Too much technical debt accumulated or the processes that make Facebook Facebook in a good way backfired in a way that made you perhaps question the validity of some of these practices.

[00:07:15] PH: One in particular comes to mind. There was this project that was released a number of years ago called Facebook Home, which was it was like it ended up being a lock screen for Android, but it was also paired with like a Facebook-branded phone from HTC, and there was a ton of work to do a project much more ambitious than that leading up to that project. Facebook's approach to that type of mobile development, I mean, kind of like really digging into the OS and building something that hadn't been built before was to effectively take the playbook for the PHP code base and apply it to like lower-level systems development on a mobile device in a market that hadn't been proven yet.

The companies couldn't figure out a way to rapidly iterate itself to a high-quality product, and that product that ended up launching was like multiple years late to market, and in my view kind of missed the market, and that's one of the reasons why it wasn't successful. The reason why it was so slow to market was just because the company – And this a common thing that like lots of startups and big companies do. They take their like founding myth or the one thing that really works and they say, "This is what makes it special. We can go and apply this to this other problem space." Sometimes it works. In this case, it really backfired and didn't work at all.

[00:08:37] NS: Yeah, I guess my example is much more personal in so far something that I personally participated in. The story of the bulk of my career is kind of the stack that led up to GraphQL. Then GraphQL, and then moving on to our native mobile clients. Meaning that building software for our native mobile client, specifically the GraphQL SDK.

We initially moved to iOS, and I was able to pair and team up with a couple of really talented iOS engineers and really have very significant positive impact on that CodeBase and

architecture very quickly and it really set us up for success on that platform for years. Then I sort of – I think I got a little too big in front of my britches and moved on to the Android CodeBase and kind of executed the same playbook and was really focused on moving quickly and kind of taking the architecture that had been laid down there as given.

Then moving quickly within that architecture without examining first principles and ended up making some fairly fundamental mistakes in the design of that architecture that it echoed for years.

Looking back on it, I should've approach that with more humility and long-term planning and understood the technical constraints of that platform at a deeper level before just kind of charging ahead and delivering near term to medium-term value.

[00:10:07] JM: One way we can look at Facebook is as its engineering culture relative to other technology companies. We've had some conversations about how Facebook contrasts with the culture of innovation at Apple, the culture at Google, other companies. How does innovation at Facebook occur and what are the ways in which other companies perhaps have merits in their innovation processes that Facebook does not have?

[00:10:44] PH: Yeah. I've been thinking about this a lot lately, and I think it's important to separate what we're doing with how we're doing it. So Facebook in the what we're doing category, this is like what product bets are we making, what sorts of markets are we going into, what features are we prioritizing, that sort of thing is pretty top-down. In a similar way that Apple from what I know is pretty top-down. There're a couple of visionary product leaders at the top. They are very, very assertive in making sure that the company is executing against that vision with the right prioritization. If you look at the success of Facebook and the success of Apple, that seems to work pretty well.

I would contrast that with what I know about Google, where Google has done a very good job of creating an innovation machine, where it's not reliant on a couple of key people that are like hyper empowered to do the innovating. They really created a process and a machine that allows individual lower-level teams and product groups to go and innovate on their own. I don't think

you have Larry and Sergey weighing in too much on Gmail, for example. Whereas at Facebook, it seemed like Zuck was approving and driving the majority of stuff that we would do.

There are two different approaches. I think the advantage of those kind of couple of visionaries at the top handing it down is that it's super organized. You know exactly what is the top priority what you should do. There's not a lot of bureaucracy around it, because it's just kind of – Because we already know what to do and it's really just about how do we do it.

At Facebook anyway, the how do you do it was very laissez faire, which I think was one of things that made it a great engineering culture. That works really well for Apple and Facebook, but it is dependent on a couple of key people, and I think that Google is probably a more resilient organization in that respect because of way that they've built that machine.

[00:12:44] NS: Yeah the one thing all add to that is that Facebook has been able to add to use the Apple parlance directly responsible individuals, DRI's, for huge swaths of their product line via an acquisition strategy.

This is mostly – To be clear, Pete and I were both engineers and engineering leaders and were not involved in the strategic command of the secret high council of products. We're more viewing the aftereffects of this. But by acquiring independent brands such as Instagram, such as WhatsApp, such as Oculus and having that separate brand, having very strong empowered leaders on top of that, like the fact that we were able to retain the Instagram founders for six years after a billion-dollar acquisition is unbelievable actually. I think that that is the way that Facebook, the company, has started to kind of scale product innovation.

[SPONSOR MESSAGE]

[00:14:01] JM: If you want to extract value from your data, it can be difficult especially for nontechnical, non-analyst users. As software builders, you have this unique opportunity to unlock the value of your data to users through your product or your service.

Jaspersoft offers embeddable reports, dashboards and data visualizations that developers love. Give your users intuitive access to data in the ideal place for them to take action within your

application. To check out a sample application with embedded analytics, go to softwareengineeringdaily.com/jaspersoft. You can find out how easy it is to embed reporting and analytics into your application. Jaspersoft is great for admin dashboards or for helping your customers make data-driven decisions within your product, because it's not just your company that wants analytics. It's also your customers.

In an recent episode of Software Engineering Daily with TIBCO, we talked about visualizing data inside apps based on modern frontend libraries like React, Angular, and VueJS. If you want to check out episode, it's available on softwareengineering.com. You can also check out Jaspersoft for yourself by going to softwareengineeringdaily.com/jaspersoft and finding out about embedded analytics.

Thanks TIBCO for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:15:37] JM: So one thing I'm trying to reconcile is what you guys just said, the top-down product direction with a sense I've gotten from other interviews that Facebook has a bottoms up high trust collaborative culture where any engineer can come into Facebook and say, "I'm going to be an influencer engineer. I'm going to build up credibility and then I must start something cool within Facebook and it's going to take off, because I'm going to marshal the resources within it."

Maybe this is a product of – I'm thinking about Facebook in different timelines. Maybe the early Facebook, there was more bottoms up and then now it's more of a top-down organization. But could you help me reconcile those two understandings of the Facebook innovation culture?

[00:16:31] NS: Sure. It goes back to the Pete's premise and preface, which is separating the what from the how. Meaning that there's top-down context setting, and context setting means what's the product strategy. It's not like you can come straight out of boot camp and then decide we're going to pivot the company.

[00:16:55] JM: We're doing dating.

[00:16:56] NS: What?

[00:16:56] JM: We're doing dating.

[00:16:57] NS: We're doing dating. Right. I graduated from boot camp and we're doing the dating app. That's not how this works. What you can do if you're an ambitious engineer, and this is all within certain parameters as you build up trust and reputation, is within the context of that overall company strategy, you have an enormous latitude as to the technical artifacts that you can produce that execute on that strategy. Within that dimension, there's tons of innovation, right?

Let's take an example of a previous interview of Keith Adams with HHVM. What virtual machine executed PHP had nothing to do with product strategy, right? Completely different layer of the stack. HHVM was extraordinarily innovative on a couple of dimensions. But what there was a mandate of is that we want to execute on product strategy X, Y, and Z. We want to do so while not bankrupting the company, and that was pretty much it. That is the context setting that allowed all these various runtimes to execute.

[00:18:10] PH: Yeah, and I think that that strong assertive like top-down – By the way, it could be very stressful. The highest priority projects would get like weekly Zuck reviews, if not twice a week. So you definitely felt this pressure to move fast and execute on stuff very quickly and at high quality. If you look at something like React, creating a JavaScript framework is like the classic example of like an engineer procrastinating on the real work to do. I don't know, at least in my experience I feel like that's a good example.

But the reason why React actually happened was because there was a guy working on the ads frontend. The most hardcore frontend username or the most hardcore user-interface in the whole product in 2010, 2011 was the flow to create ads on the platform. Obviously, like as leading up to the IPO, getting that thing right was super important. That team had a lot of pressure, and like delivering new features on that thing got slower and slower the more complex it got.

there was very, very strong top-down pressure on this thing to behave a certain way and engineering and the people figuring out how to do it said, “Listen, we have to make this investment in some frontend infrastructure to really accelerate this work.” That wasn't like they sat down in a meeting and like created a JS framework initiative. It was like one engineer, Jordan Walke, nights and weekends kind of hacking on this idea that he then kind of was able to successfully sell to the organization as what we should actually do.

The only reason why that happened was because Jordan was working on something that was headed down top-down, really internalized how important it was and was able to tie that work up to those really, really clear goals that the company had. I don’t know. Is that a fair assessment of how React ended up happening?

[00:20:08] NS: Yeah, and I just love the JS framework as procrastination line. That resonates.

[00:20:15] JM: Speaking of open source, Facebook's open source history is distinct from how open source has manifested other companies. Why did Facebook's open source culture end up the way that it did?

[00:20:32] PH: I think Nick's got some great perspective on the open-source strategy of the company, but I can start framing up, that from my perspective, some Facebook open source – I think the vast majority if not everything that Facebook open sources is of a high quality. I think everything is like technically pretty excellent.

But it's very clear that some projects that Facebook has open sourced have become wildly successful. React, GraphQL, Hip-Hop, when it was originally open sourced was like highly successful as well. Then there have been some projects that have been colossal failures, like the first release of Relay, for example, was very technically excellent, very good for Facebook, but like nobody could figure out how to use this thing.

Then there were some projects that were released that were like kind of failures and then they really iterated on it and got it back to healthy, like Jest is a good one. Oh, yeah, and then there’s something like Flow, which was really, really popular and then kind of tapered off.

The point I'm trying to make here is that Facebook's projects are actually all over the map, and the reason for that I think is because, at least based on my own experience with React, it's driven by individuals. If the individuals that want to do this thing and get this thing open source, if they value documentation, if they understand where the community is and they can make the internal Facebook thing, meet the community where they are, then it will be successful. But if it's not or if there's just an impedance mismatch between the two, then it won't.

[00:22:07] NS: Yeah. As Pete alluded to, I have many thoughts on this subject and is probably worthy of an entire episode with lots of different people, because I think it's not just a Facebook issue, but an industry-wide issue, and that is precisely what is a relationship between venture backed and profit-making enterprises that produce infrastructure for their own business requirements? Then what is the incentive alignment with their projects that get open sourced typically driven by the individual engineers who are responsible and participating for building those open source projects.

That's kind of I think the general industry-wide problem statement that I still don't think there's a crisp articulation of. Now, with Facebook open source strategy, first of all, there is no strategy. There is a general feeling, I would say, from the top, that it's just kind of the right thing to do and there is a pay it back sort of mentality in terms of, "Listen. We know the fundamentals of this business were built on an open source stack Linux, the LAMP stack, initially, right?"

There's kind of the notion that it's like the right thing to do. Then there's kind of this random walk aspect, ironically, like Jordan Walke, where engineers just kind of have done it and then the breakout successes, in particular, React, which is a sui generis rocket ship project. GraphQL is kind of a baby and React is like the BFR, if you know that means.

Schreop, the CTO, Mike Schroepfer, was always very supportive of these projects, he even said to me, he was like, "Listen. We're launching probes, drones in the low orbit that are shooting lasers at each other and the only thing anyone wants to talk to me about outside the company is a God damn JavaScript framework."

There's like this post hoc justification for the investment that is kind of in terms of reputational brand aspects and for recruiting. There's that aspect, but in reality, beyond this kind of difficult to

measure sort of engineer NPS score, net promoter score. I'm just appropriating marketing lingo for this. It is very difficult to tie the business metrics of Facebook to the underlying health of its open source projects.

Connecting those dots is at best through this intermediate goal. I think the perfect example of that is the Flow project that you mentioned, that Pete mentioned. Let's take this in particular. Why is Flow not very popular? It's not very popular because there is a competing project called TypeScript out of Microsoft that Microsoft has a business proposition for supporting.

TypeScript is a developer tool. Microsoft is a developer tools company. They have assigned senior top talent to that project. The leader of that project is Anders Hejlsberg, who might be like – I don't know if he's best – At that point, he's one of – In a rare echelon of one of the best language designers in the world and someone I personally look up to. They have massive investment around both VSCode and TypeScript and this huge ecosystem.

Even if there's like some particular aspects of the Flow type system, which are better and like I think there are actually, but it doesn't matter, because the ecosystem around TypeScript is so strong. I don't know what the end result of this little soliloquy is, but I think it's like it's not a Facebook issue.

I think it's an industry-wide issue and they're still figuring out and everyone's figuring out, because like, "Hey, if our engineers are distracted with maintaining this open source community and serving customers of their stuff or even our competitors, why are we doing this? Why aren't we focusing them on something that can clearly be tied to the business objectives of the organization?" I think that's more about setting context for what the issues are rather than proposing any sort of coherent philosophy around it.

[00:26:44] PH: One thing I want to add though, which is its disorganized. There is a certain set of checkboxes that you have to the check before the open source team will let you release something. You have to have somebody who is responsible for maintaining some SLA on pull requests and a plan for making sure that changes get synced out and all that stuff, like being a good open source steward requires. But it is like basically chaos. It's like some projects

randomly get like shot out of the Facebook codebase and some don't just based kind of on who is involved and what they're working on.

I think that if we want to look at – We could look at Facebook through this lens where a lot of people would come in and say it's pretty disorganized actually. They would look at – On one side of the coin we can say, Yeah, there's a lot of – With respect to how we do it, there is a lot of freedom for engineers to figure out the best way to do it. There's not a lot of like mandates for how to work.

But on the other side of that coin, some people would call that disorganized and some people would criticize that and say, "Hey, listen. The people with the loudest voices are making the decisions implicitly rather than explicitly."

I think if we're thinking about what are some things that Facebook might not do well is you might ask yourself, "Are those decisions that are being made the right decisions or not?"

[00:28:09] NS: This is just somewhat related, but I feel morally compelled to discuss this issue, which is to build on Pete's point that it's a little chaotic and there's not like some super coherent overall strategy. One thing that the lack of clarity around this causes a lot of confusion about licensing and patent issues where when we open source something, there is often a lot of theorizing that these open source technologies are somehow some long-term plan to do, like an intellectual property power grab in the industry and so on and so forth, which can often cause some like pretty hostile behavior from people who object to some of the licensing aspects of it.

[00:28:53] JM: The WordPress story.

[00:28:55] NS: Right. This is just purely a request to those who care deeply about this issue, and I understand that you do, is please understand what's actually going on here, which is you got some engineers within Facebook who are trying to do their best and they're really passionate about this stuff and they are doing with their own internal institutional cleavages here. Then when you are assaulted for this externally, it's like psychologically very confusing as to what's going on, because it's like, "Hey! I'm just trying to produce some tech."

The leaders in the legal counsel involve are just like naturally conservative in terms of not exposing the company to intellectual property nuisance issues, for example, and there is no conspiracy. The conspiracy does not exist. As the person who is effectively kind of primarily responsible for convincing the right people to open source GraphQL, there was like no conspiracy. We just thought it was the right thing to do. Please approach those subjects with empathy and understanding.

[00:30:00] JM: Yeah, that's kind of the story of Facebook getting criticized recently that you calcified if pretty well. I mean, the open source issues with WordPress was like a drop in the bucket, but if you compare that how Facebook has been criticized more recently, I think it's a lot of what you said maps on to that as well.

Okay. So coming back to the disorganization thing, I think the people who the disorganization might hurt the most are people who are either new to Facebook or struggling to figure out their place at Facebook. We have this interesting conversation that was regarding the Tom Occhino interview, and because I was very impressed with Tom in terms of how he spoke about management. The thing that I was curious about was like can use scale engineers being satisfied with their work?

When I think about “the chaos” of Facebook's engineering environment, I kind of contrast it with the companies that I worked at when I was a software engineer. I worked at eBay. I worked at Amazon. I worked at a couple of other companies. In all of these jobs, my work was very prescribed to me. It was laid out in in clear terms, and that actually worked very well for my coworkers. It did not work well for me. I wanted freedom. It wasn’t that I didn't want to do my job. I was working like longer hours, but I need to have an outlet. I needed to have the ability to explore and to do things that that might've felt chaotic to other people in the organization. In that sense, the disorganization kind of appeals to me.

I guess what I'm trying to get at is I feel like engineers should have both the requirements of their job, but they should also have this latitude to explore and to do things that they like within the organization. My sense is that Facebook has somehow captured that lightning in a bottle and scaled it. I do believe what Tom says that he is able to satisfy the desires of those

Facebook engineers who are seeking some kind of internal validation in terms of the projects that they're doing.

I don't know. You guys are both managing people at this point, and I'm wondering how you balance and how you think Facebook has successfully scaled the necessity to do kind of boring work with the joys that can come from exploring the chaos.

[00:32:58] PH: That is really interesting question, and I think I actually talked about this with Nick a bit, and I'm going to steal a couple of his talking points. But the first thing that you got to understand is that it's really hard to get at Facebook. You have to be like really good on a number of dimensions. The stereotype about intervening at tech companies is that it's is all about writing code in the interviews and whiteboard coding and algorithms and stuff.

Yeah, that's a component of it, but there's a lot of screening as well around like can you architect the system? Do you know when you're actually doing something that's the right – Do you know when it's time to invest in an architecture versus just delivering on product features or are you going to be creating new technical debt? That type of thing.

Then there's also mission alignment and being able to communicate and being able to understand what the company's goals are an balance those against the technical objectives too. First of all, the engineers that make it into the company, you should be able to have like kind of a mature conversation with them and be like, "Listen, this is not the most interesting thing in the world, but we need to work on it for a little while?" They should be able to have the patience and not be like a diva about it.

The other thing is that low performers do get fired easily at Facebook. I forgot what the stat was, but they definitely performance manage people and fire them if they're not doing well. But it's not like you have one bad week or one bad quarter and you instantly get fired like. There is this notion that like we want to find the right place for someone to be successful. One of the things that really works for Facebook is that the product surface areas huge and every single one of those products is super scalable or at like really high scale.

You can always kind a like find a new products surface area or a feature to put someone on or you scale up the system, now there's this other thing that's breaking down, go work on that. I think that like maybe Facebook wouldn't work as well if the product was boring or like it wasn't very broad, or it didn't have interesting technical challenges.

[00:35:08] JM: Or wasn't growing.

[00:35:09] PH: Or wasn't growing. Yeah. I don't know. Nick, what do you think about this?

[00:35:12] NS: I think it's a really complicated subject. It's super context dependent. That all being said, I think one thing that helps is how you – There's a certain percentage. I think some of the stuff we're focusing on this section of the conversation is like there is this like any engineer organization, there's this stuff that you don't want to do, and it needs to be done.

If the premises is like, "Okay. You allow the engineers to choose all their own stuff and then magically everything is going to get done." But that's not true. How do you factor their goals and objectives that satisfy both of these constraints? They get to do directionally what they want to do or what they're passion about, should I say. That's aligned with the objectives of the company. How do you make sure the kind of the odds and ends get done?

I think to me the ideal way of accomplishing that is giving an engineer a sense of ownership over a high level objective, and then in order to achieve that objective, they are responsible for – It's only going to work if they do that if they spend 20% of their time doing stuff that they might not necessarily want to do.

Really imbuing them with a sense of ownership over the end goal of what they're doing. It's like, "Okay. You're working on this framework. Your job is to make that successful." In order to do that, yes, you get to do the fun stuff of like architecting things on whiteboards and like sitting back and beard stroking or whatever you like to do. But in order to do that, you also have to write documentation. You also have to respond to bug requests. You have to do all that stuff.

Then what you want to make sure is that engineers – Arturo Behar, very successful, very effective manager of engineering ICs. He was, "Listen. Tike there's going to be some

percentage of time that you're doing stuff you don't want to do. The key thing for sustainability is to make sure that N% is at a sustainable level. Maybe you spend 80% of your time doing what you like to do, or maybe 60%, and 20 things of the things that you tolerate and 20% you don't like to do." It's up to you to own your experience so that you can like factor your stuff that makes you accountable to your teammates that you're not offloading all the crap work to them but also like you're at a sustainable level of production. I thought that was very wise and practical advice.

[00:37:42] PH: I wanted to add one other thing, which was like the career ladder at Facebook. I think there technically was like a rubric for an E4 to an E5 to an E6, and it probably had all the standard stuff that every other company has. But, largely, Facebook had these posters all over the place that would say things like focus on impact, and impact was really the number one thing that anybody cared about. So whether your project was easy to do or was impressive technically or not, didn't really matter as much as the impact.

I think what it often comes down to I found in my experience anyway is the stuff that people don't want to do tends to have a pretty high impact score if you're asking them to do it. If you've got these posters all over the wall that say impact, impact, impact and you recognize high impact, whatever it is, and you have a pretty expansive definition of what impact is, you can generally frame it up in that way.

If you give someone 100% of the bad work for their multiple quarters, they're probably going to quit. But I think if you can incentivize that a little bit by talking about impact. The one other thing that came to mind when you were talking, Nick, was going back to the – Making a JavaScript framework as an example like procrastination and beard stroking and stuff like that, at Facebook you would basically get laughed out of a room if you proposed some sort of frameworky thing and you had no examples to back it up from like your day-to-day work.

One of the reasons why there are like a couple of like – There're many Facebook open source projects that are wildly successful. I know that we just talked about kind of the hit or miss nature, but like there's a lot that are really successful. One of those reasons is because like anything that would've been architecture astronauted would be killed instantly, and it's much more like – Yeah, I was working on ads for the last year and I found this problem and now we've got React

as the solution. Hey! We're building these new iOS and android apps and like we're going back and forth to server all the time. So we built the solution.

Then when you go and you present that framework to the next team to go and try to get them to use it, they're going to ask a bunch of questions about, "Hey, I've got this problem in my day-to-day. What do I do?" You can point back to evidence. If you can't do that, no one's going to take you seriously. I mean, it's almost like starting a company. If you don't have any customers or case studies, the customer will be very suspicious of the thing that you're trying to sell them.

[00:40:16] JM: Getting to questions closely related to infrastructure, one subject we've had a number of conversations about is this mono repo monolithic software architecture and the tooling and unique processes that Facebook had to build around that. I guess I'd like to know more about the strains that the monolithic mono repository puts on an organization. What kinds of tooling needs to be built? Why this is so rare to see in the industry relative to the epidemic of the microservices architecture.

[00:40:58] PH: It is an epidemic.

[00:41:01] NS: I concur, but we won't go into that today. I think one of the primary reasons that otherwise talented engineers who can – Or brilliant and otherwise be very successful at other companies would come into Facebook and then churn out quickly of their own volition, is they couldn't deal with this like level of controlled chaos that occurred within the company. I think the control chaos, the product codebase mono repo is a great example of like muddling our way through that and figuring that out.

I think you have to like, for example, within the mono repo and within the product codebase, you have to think about ownership in a different way. One of the reasons why microservices, for example, are super clean in terms of ownership, is that there's a process boundary around team organizations, or like bug attribution is easier and like there's a very clear ownership of like we own this part and you own this part.

When you're in process with other people's code, when you're in the same codebase, it gets way more muddled. The ownership structure needs to be less like Stark dividing lines and more like overlapping curves of responsibility, right?

I'm visualizing it to you because we're in the same room. For example, I remember in the earlier days with the frameworks that I was working on and the feed team, and there wasn't a strict dividing line. We overlapped. We overlapped with each other. Me and the team I was on was primarily responsible for this core framework, but we have latitude to reach into the feed codebase and maybe even change their codebase in order to implement a framework feature.

On the same token, they have latitude, because we have a trust relationship to go in and propose features and even just do them, right? Now, obviously, we would tend the core and team would tend to change that more, and the feed team would tend to change feed more, but there is this kind of like overlapping nature to it. Navigating that is both a technical concern as well as a relationship concern and a trust building exercise. It's just more complicated. That's kind of like one dimension that strikes out at me as making it more complicated and difficult to manage and scale a product codebase.

[00:43:34] PH: Another thing I would add to that is about tooling. Whenever your team hits a certain size of engineers committing to assist them – I mean, it's not just one codebase. It could be like different services talking to each other. You need some degree of tooling to ensure like high-quality and that people can work somewhat effectively whether it's a microservice or a monolithic architecture.

I think you probably need about the same – There're probably some ratio lines of code of internal tools to like product code. I think is probably like pretty constant for whether it's microservices or monoliths, but I think in 2019, there is a lot of vendors that will sell you a lot of great tools to manage a microservices deployment. All is distributed, tracing stuff, like LightStep is really snazzy, and Facebook had to build a lot of that stuff. The release engineering team had some magic tool that would look at a stack trace for a new bug that would show up during a canary and then like attempt to find the right person to triage that bug to. Whereas in a microservices deployment, you have an owner's file for that service and you can triage it to the right on-call.

It requires a tooling investment, and I think that the tools that you have for monoliths are baked into the frameworks like Ruby on Rails. I think Ruby on Rails is just not designed to scale past like tens of engineers. We can dunk on Ruby on Rails for the rest the interview if you want. I mean, Twitter is a good example of that like tried to scale Ruby on Rails up.

I mean, this was years ago, but it's still the same thing. It's like it's dynamically typed. Everything happens at runtime. There're all sorts of mix-ins and stuff and it's very difficult for you to like read the code statically and figure out what's going on.

I think if the open source frameworks were better, like monolithic frameworks were better at scaling to larger teams, we might be in a different situation today. Yeah, I don't know. That's kind of my perspective on that.

[SPONSOR MESSAGE]

[00:45:38] JM: Better.com is a software startup with the goal of reinventing the mortgage industry. Mortgages are a $13 trillion industry that still operates as if the Internet doesn't exist, and better.com is looking for engineers to join the team and build a better mortgage experience.

The engineers at better.com are attacking this industry by bringing a startup approach into an industry filled with legacy incumbents. Better.com automates the very complex process of getting a mortgage by bringing it online and removing the traditional commission structure, which means that consumers can get a mortgage faster, easier and end up paying substantially less.

Better.com has a modern software stack consisting of Node.js, Python, React, TypeScript, Kubernetes and AWS. They iterate quickly and ship code to production 50 to 100 times every day. Better.com is one of the fastest growing startups in New York and has just announced a series C that brings the total funding to $254 million.

If you're interested in joining a growing team, check out better.com/sedaily. Better.com is a fast-growing startup in a gigantic industry. They're looking for full stack engineers, front-end

engineers, data scientists. They're looking for great engineers to join their quickly growing team. For more information, you can visit better.com/sedaily.

[INTERVIEW CONTINUED]

[00:47:17] JM: There's a phenomenon that you call Facebook Moore's law, which is essentially the idea that the product engineers at Facebook, they are not given quotas, they're not given constraints really on the resources that they can consume. They can kind of go hog wild consuming resources. I mean, assuming that they actually need those resource or are making good use of them and then the infrastructure team just kind of figures it out, figures out how to scale in response to that. Maybe this is this is like less applicable now, because I'm sure the Facebook infrastructure team is quite well scaled, or maybe not. I don't know. But at least this was true when you were there. Can you tell me more about the Facebook Moore's law?

[00:48:03] NS: So isn't a term that people really talk about, but as I was thinking it through – I listen to Raylene's interview, and she was talking about how every engineering all hands, they would hear, "Hey! Our infrastructure got faster. It got cheaper. It got easier to use." She talked about how that's a pretty rare thing in a tech company.

I started thinking about the implications of that, and at Facebook, we would roll out pretty major features and we wouldn't have a separate quota for those features. Most tech companies, if you want to go write hundreds of gigabytes or terabytes of data and do some database tier, you have to go file an infrastructure ticket. Make sure there's enough capacity. Make sure there's an on-call register and all sorts of stuff like that.

As a product engineer at Facebook, at least in that era, I think I talked to an infrastructure team once and it was like, "I had roll this thing to production for 1%," and they said – They just send me an email, they're like, "Hey, do you know how big this will be?" I said, "It's about going to be this big," and like that was the extent of it. I wasn't blocked on that conversation at all.

I attribute that to, again, like people like Keith Adams and other people in the infrastructure team, stuff would just get faster and cheaper. I don't actually know how infrastructure did it. I think it'd will be really interesting to talk to more or those folks and see if this was like an

intentional relationship that they want to have with product engineering where it's like your job is to just make the infrastructure totally transparent to the product engineering team and let them iterate without worrying about it. But it worked really well.

[00:49:41] NS: I have a lot to say about this. First of all, I likewise took it for granted while I was a product engineer at Facebook. I didn't realize how special that dynamic was and I often – Not often, but I do have some time some guilt about taking that for granted despite being like, "Of course, that's how it works." No. That's not of course how it works, and it was definitely an explicit decision by the infrastructure teams in order to operate in that way, and they did extraordinary, extraordinary work.

This is one of those things where you want to build a technical architecture that appropriately matches the domain of software that you're building. When we talked about a monolith, we were talking specifically about application code. The infrastructure team builds hundreds of services that have very well-prescribed APIs and contracts that do go over our process boundary. So they very appropriately have their own teams that execute and there is strict ownership, or much stricter ownership I should say.

Let's make a very concrete example of what he's talking about, what Pete's talking about with Facebook Moore's law. Actually, like every quarter, you would read some update from the Hip-Hop HHVM team that would be like, "Yeah, we implemented such and such a feature." So we had a goal of reducing the CPU load across our entire server fleet by 8% and we actually did it by 10% and you're just like, "That sounds like a lot." Then it would keep on happening every single quarter.

So that's what he means by Moore's law, where it would compound quarter after quarter after quarter, and the infrastructure teams would just be delivering this stuff. It's just like it's incredible to work on. Then they just made a commitment that they built these centralized pieces of infrastructure that serve the product engineers, and the product engineers could just go and do their worst and they would figure it out, and it was a remarkable dynamic.

The scale numbers are insane. We had this system called TAO that was effectively a distributed key-value store, plus we had – It was what stored our application data and the graph of data in our system. The scaling requirements of that was crazy.

I remember onboarding or talking to the parse team about when they were relatively new to the company. I was explaining TAO and at some point they were like, “Wait. What is the QPS of this thing?” Then I knew that at that point. It was like, “Yeah, it's 2 billion queries per second in the worldwide fleet.” It's probably like 5X that now and it just like we just kind of took that for granted, which is fairly crazy if you think about it. Bravo to the infrastructure team. Let me reemphasize that they do have a services architecture and it's appropriate for their domain.

[00:52:38] PH: But I want to put an asterisk on that. The reason they have a services architecture is for basically like resource isolation, independent scaling, error boundaries, stuff like that. They don't have a microservices architecture or services architecture because they don't want to talk to each other or work in each other’s codebase. It’s still services architecture in a mono repo.

[00:52:59] NS: And it’s one codebase. Yeah.

[00:53:00] PH: Yeah, it’s one codebase. There’s actually a separate mono repo for that, but it's still the same style of working, but it is deployed independently. One thing I was always curious about, and I don’t really have the answer for this, is at Facebook, if you look at Hip-Hop, right? So moving from open source Zen PHP to Hip-Hop, there were certain features that were very difficult, if not impossible to implement in Hip-Hop PHP. As a product engineer, I don't ever really remember being asked to refactor my codebase because of someone else's infrastructural change. I think what you will find in a lot of other companies, and I think you had a podcast with somebody from Google and it seems like the same thing at Google where it’s like, “Hey, I am re-factoring my code. I'm deprecating some feature.” Then you go to all of the customers of that feature and you ask them to refactor and get on the new API. That like literally never happened at Facebook. It was all just done for me. Is that your reader as well?

[00:54:04] NS: Yeah the HPHP team definitely refactored the product codebase itself in order to implement their features.

[00:54:09] PH: Was that like a thing that was consistent across all the different pieces of infrastructure or like was that like a Facebook cultural norm or was that just something that didn't happen that often?

[00:54:19] NS: I think so. The other thing you would do is do a kind of in sequence where you would come up with a higher level abstraction and then ensure that that's stacked well with the future direction of the underlying infrastructure. So as engineers migrated to that new thing, it would naturally be utilized, and then presumably the new underlying infrastructure would deliver a lot of value on some dimension, therefore increasing the incentive to adopt that new centralized abstraction. By working in concert with the input teams, you can kind of like accelerate an adaption path.

[00:54:49] JM: By the way, at Google, I think what Sierra referenced in that interview with her was actually that she was giving the example of updating like a GRPC or a proto-buff client or a compile or something and she said that actually if you do that, you actually go to all the customers and you refactor it for them in the mono repo. In the mono repo environment, you actually do it for them.

[00:55:16] NS: Google has a mono repo and the amount – I mean, you did an episode on it. The amount of investment they put into that thing is bananas, like totally bananas.

[00:55:26] PH: Did you listen to Sierra's podcast?

[00:55:29] NS: No.

[00:55:29] PH: It's good.

[00:55:31] NS: I've heard her talk publicly. Actually, I was at a conference and someone from Google gave like an hour-long presentation about their mono repo technology before I went on stage and I was like – The thing I'm talking about is not nearly as cool as that.

[00:55:46] PH: Bazel is pretty good too.

[00:55:48] JM: Not to go down that route too deeply, but do you guys have perspective for like is it possible to get the necessary tooling to move the industry in the direction of mono repos?

[00:56:02] NS: I believe so. To me, the awkward stage of mono repo management is this kind of middle phase where you've scaled enough where a mono repo starts to reach its edges of either performance or some lack of other tooling and you're some medium-size company. In order to make the mono repo work, you're going to have to invest a significant amount of tooling. If you're unwilling to do that for understandable reasons in the context of your business, then the only option is to break it apart.

I think it's worth kind of calling that out, and maybe there is a missing tools company out there that supports that kind of middle ground. but the value of that has to be proven, right? Meaning that people aren't going to adapt that way unless like they believe it's valuable. So there's the chicken and the egg problem there. I think it'll be a challenging business to build actually.

[00:57:03] JM: So let's revisit this question of Facebook's engineering culture relative to other engineering cultures. What could Facebook learn or what do you see as valuable in other mainstream large engineering cultures relative to Facebook when we contrast Facebook with the Google culture, the Amazon culture, the Apple culture? What could Facebook learn from these other organizations?

[00:57:32] NS: The thing that comes to mind for me, especially in the modern context. where we've reached the limits of scaling companies in one geographic location partially because the particularity is the Bay Area, and housing prices, and geographic constraints even, is that I don't think Facebook is much had a culture writing things down as much as other companies.

In order to effectively work at multiple geography scale well, I think one of the ways you can do that is have a culture of writing things down so that you write down all decisions, you write down why they were made, the tradeoff, so that everyone feels included in that decision-making process. I think that Facebook has tried to do that as time has gone on, but it's still like that was not the way it worked in its formative years. I think that scaling Facebook geographically has been a challenge.

We definitely have teams in the different offices that interface very well. They typically do so across API boundaries where they can interface well in a way where they don't have to necessarily broadcast their decision-making processes as much. But I think that other companies do that well and serves them well and gives them a lot of flexibility in terms of having remote work or many more distributed geographies.

[00:58:57] PH: Yeah. Yeah, I would definitely agree with that. I would say Facebook like almost actively fought that for a long time. Zuck, for a long time, refuses to open an SF office because he wanted everyone co-located for the very good reason that co-located people generally work better and share information better more effectively than people that are remote. If you want to be remote you have to really work at it.

I definitely think that the lack of writing things down both kind of hurts being able to collaborate between offices and also it's a little bit more difficult or it's way more difficult to understand the historical context of stuff. When I came to Twitter I was shocked to how good Twitter was at this stuff. Almost every meeting would have an agenda, a shared Google Doc, commenting back-and-forth on the Google Doc. The default when anyone walks into the meeting room is to turn on Google Hangouts, which is really cool. Then there're technical design documents for everything.

For me, have very mixed feelings about these large detailed technical design documents, because the Facebooker in me says, “Listen, like we could have implemented this thing by the time the technical design spec was written and approved.” At the same time, I can just sit on my laptop and read the entire history of all Twitter's architectural decisions because they’re all in one place easily digestible with the primary source discussion in there. That is something – I think there's some middle ground there that Facebook could learn from and maybe they have. I don’t know. That seems like they may have done this since

[01:00:25] NS: Yeah, I'm I was there later than you were. So there's definitely been investments in the remote aspect and working – Facebook has a few primary engineering centers; New York, Seattle, Menlo, now San Francisco now that’s growing and London. It's not massively remote, but there are distributed geographies with time zone issues.

The amount of internal investment and the scheduling and the videoconferencing software is extraordinary. Facebook does not use Google Hangouts. Facebook has its own proprietary stack, and I believe proprietary hardware now for only its internal usage. Totally wild. The scheduling tool that we used was very good. You could like configure, select attendees, select like, “I want every other week here,” and then you would select it and it would be a constraint solver and be like, “Okay. We can use this room and this geo, this room and this geo,” and then it would align all the schedules and then it would figure out like, “This works, except for this one outlier case where there is one meeting that overlaps six weeks out.” You have to move that one and there’s like a tooling flow to do that.

What I was talking about in terms of writing things down and the remote stuff was in terms of the initial culture formation rather than the investment and internal tools. It's now recognized as an issue that requires significant investments. There’re massive resources behind the tooling around that. I think at this juncture, I’ve been gone for the company for two and a half years. Pete? What? Five or something? Four? This would be like something where someone who is actually there maybe who’s working from London could speak to the challenges of doing that.

[01:02:08] PH: Actually, I think you’re right. I think it’s five now. The more I do manager stuff, the more that I realize that like teams and people, like there's a lot of analogies to like distributed systems. Facebook's approach to remote work historically was kind of like vertical scaling or vertical sharding. If you have – For the engineers listening, if you have – You start out your app with one big SQL database that holds everything or one big whatever database that holds everything and eventually like the first bottleneck that you hit is going to be a like one table or two tables for one specific application. Rather than build out this big infinitely scalable horizontally sharded thing, you basically will normally like take out that one table and move it to a separate database server, because it's really easy to do and then you can kind of scale that one independently vertically.

Only when you get to hyper scale and you really need to systemically solve it do you move to a horizontally sharded model where you have a fleet of database servers and all tables are kind of striped across every node. Facebook's approach to remote work – Facebook had remote offices for a long time, but it was that vertically sharded model where like the Seattle team would be

working on I think it was like or some like modernization thing. The Seattle office was like basically dominated by monetization work.

In the London office, I forgot what they focused on, but they focus on something in particular. That works for a while, but when your eng organization is like Facebook and it's like predicated on the idea that you can move people around really easily and it's a little bit chaotic and really agile, you need to be able to like – In my view anyway, figure out how you're going to have people in London collaborating with people in San Francisco and Seattle on the same project. Otherwise, it's not going to scale.

[01:03:53] JM: Doing these interviews and listening to the interviews with other people, final reflections on downsides of the Facebook engineering culture. Things that people listening to this should take away. They're hearing lots of great things about Facebook engineering. What's the bear case?

[01:04:14] NS: Well, to summarize the bear case, I think you need to think about how – Like what the biases are in your organization and make sure that they're the right ones. If they're not actively work to create kind of new incentives to bias the organization the right way. Facebook is mostly a large-scale consumer application that is written on a stack that is fast and easy to deploy.

So the nature of that business is there're lots and lots of competitors and the cost of an error tends to be pretty low relative to some other stuff. You want to create an engineering organization and a set of values in a culture that biases towards rapid iteration and moving quickly. There are lots of companies that don't fit into that or aren't biased in that direction or don't naturally need to move in that direction. I think that we all should strive to iterate as quickly as possible, but sometimes your problem as a company is we're not moving fast enough. Our quality is too low.

So you need like really look critically at your company and say, "Are we biasing towards being too slow or are we biasing towards lower quality?" Then set up some stuff to combat the bias. In my view as a company gets large, the louder and more assort assertive voices in the room tend

to be the ones telling you all the risks and not all the rewards, because the risks are easier to articulate and the rewards are usually unknowns.

I think one of the takeaways from Facebook is that if you try to change the math on that a little bit and say, “Hey, listen. What if we prioritize moving faster? What would we need to do in order to move to that direction?” There are things from the Facebook playbook that you can take to do that, but I don't think you want to take all of it. I don't think that the Facebook playbook will work as is for everybody.

[01:06:07] PH: I guess I'll have a closing thought, which isn't directly answering your question, but more of kind like summing up what we’re talking about during the series a bit. The one thing I would say is Pete and I and I think people you’ve interviewed get very excited and wrapped up in what they're talking about. Then it ends up being kind of a ra-ra pro-Facebook fest.

But when he stepped back and think about it in a more kind of controlled manner, what I would say the prime lesson I've learned from this as well as talking to other people and getting a wider angle lens in the industry is that there is no best engineering culture. There is a set of tradeoffs that you make and you design the culture for the domain of the problem that you’re trying to solve, as Pete said.

This series is not about saying that you should take every single piece of propaganda that we've uttered, swallow hook line and sinker and then apply it directly to your engineering organization. The point of this series is that we think we’re presenting a difference set of cost and benefits, which is not frequently articulated in the industry and should be considered when you're designing your engineering culture to target an organization.

We've gone through this quite a bit. There's clear costs and benefits where you kind of can – You can get increased velocity but at the cost of chaos, which I actually think is not actually a technical downside, but an organizational kind of psychology downside. I don’t think everyone would be successful as an individual executing with that environment and it can be stressful. Just as an example.

As Pete said earlier, where things go wrong usually is people take their previous dogma, attribute that to their success and then apply it to a different domain where it doesn't fit. I think Facebook does this. I think Apple does this for sure. When Apple uses its kind of founding mythology in high-quality client software and applies it distributed systems and cloud services, it just doesn't work as well.

Instead of us blindly taking the cultural values of a particular organization, survey them and then think about what went right and what went wrong and then design your new culture, design for your target domain. Then also keep in mind that you know with the Facebook thing, maybe some of the reasons that we think Facebook was successful was in fact not why it was. In fact, the dominated thing was that the business was growing and so successful that we could tolerate and absorb these massive mistakes. I think that's always good to keep in mind.

[01:08:53] PH: Yeah. One other thing I would just add is a lot of the things that Facebook did are package deals I want to say. If you don't have really, really, really good internal tools and you don't really invest in that, you can't really move people around teams super flexibly. If you take one of these practices and apply it to your company, you move people around all time, but you're unwilling to allocate time and resources to building out a good internal tools. It's not going to work.

If you take a look at a lot of these practices, that sound crazy when. You view them in the context of all these other things that kind of sprung up around and it starts to make sense. I think Facebook was really good at growing this culture of engineering and the tooling and infrastructure around it to support it and it might be difficult to replicate that at another place.

[01:09:46] JM: Okay. Nick and Pete, thanks for coming back on the show, you guys.

[01:09:49] NS: Thanks for having us.

[01:09:50] PH: Thanks.

[END OF INTERVIEW]

[01:10:00] JM: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[END]