EPISODE 10

[INTRODUCTION]

[00:00:00] JM: Facebook has released open source software projects that have changed the industry. The most impactful projects to date are the React frontend user interface tools, ReactJS and React Native.

Before React became popular, there were multiple competing solutions for the dominant frontend JavaScript framework. React became the most prominent because of its invention of JSX, it's one way dataflow and the strength of its community.

The React community is led by Facebook engineers. Facebook has a full-time team dedicated to improving React. Facebook also has the benefit of seeing the hardest problems in web development ahead of the rest of the industry since Facebook is always pushing the leading edge of what web software can do.

As React evolves, Facebook itself is the first user of updates to the libraries. Engineers on the React team will proactively update the applications on other teams with new React code. In this way, Facebook's React team is similar to a platform engineering team, except the open source community also benefits from those improvements.

Tom Occhino is an engineering director at Facebook. He joins the show to discuss his 10 years of experience at the company and his role today as a senior leader in charge of open source developers.

We have new apps for Software Engineering Daily. They're available on the Android and iOS App Stores. These apps are a great listening experience for Software Engineering Daily. They have all of our old episodes, including all of her older Facebook episodes about the company and its culture and its engineering practices.

We have social features so you can comment on episodes and discuss them with members of the community, including me. I will be commenting on each episode going forward for a while.

© 2019 Software Engineering Daily

So if you want to have a discussion, you can jump in to the app or jump on the softwaredaily.com and share your thoughts, discuss the episodes, and you could become a paid subscriber to get ad-free episodes at softwareengineeringdaily.com/subscribe.

Also, FindCollabs is the company I'm working on. FindCollabs is a place to find collaborators and build projects. FindCollabs is also having an online hackathon with $2,500 in prizes. If you're working on a cool project or, really, any project, or you're looking for programmers or collaborators to build a company or start a project or work on music with, checkout FindCollabs. It's a place to find collaborators.

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

[SPONSOR MESSAGE]

[00:02:53] JM: When you start a business, you don't have much revenue. There isn't much accounting to manage, but as your business grows, your number of customers grows. It becomes harder to track your numbers. If you don't know your numbers, you don't know your business.

NetSuite is a cloud business system that saves you time and gets you organized. As your business grows, you need to start doing invoicing, and accounting, and customer relationship management. NetSuite is a complete business management software platform that handles sales, financing, and accounting, and orders, and HR. NetSuite gives you visibility into your business, helping you to control and grow your business.

NetSuite is offering a free guide, 7-key strategies to grow your profits at netsuite.com/sedaily. That's netsuite.com/sedaily. You can get a free guide on the 7-key strategies to grow your profits.

As your business grows, it can feel overwhelming. I know this from this experience. You have too many systems to manage. You've got spreadsheets, and accounting documents, and invoices, and many other things. That's why NetSuite brings these different business systems

together. To learn how to get organized and get your free guide to 7-key strategies to grow your profits, go to netsuite.com/sedaily. That's NetSuite, N-E-T-S-U-I-T-E.com/sedaily.

[INTERVIEW]

[00:04:44] JM: Tom Occhino, you an engineering director at Facebook. Welcome back to Software Engineering Daily.

[00:04:50] TO: Thanks a lot for having me, Jeff.

[00:04:51] JM: One of the themes that has come out of the other interviews with Facebook engineers that I've done is the impact of the company's shift from desktop to mobile. This was around 2008, 2009 and the ensuing years after that, and you join in 2009. Not long after the effort to focus on mobile had started. Tell me your memories of that shift to mobile.

[00:05:19] TO: Yeah. Actually, I think there was a couple of years where our shift to mobile hadn't kicked off yet that I was already at the company. So, we had started focusing on it, but not really an official sort of companywide capacity. For my first couple years here, it was sort of – There was a mobile team. It was a pretty small team. I remember the individuals on it and we worked on mobile web, and we had a couple of engineers that as soon as the iOS platform came out and Android started to become a thing, we had a couple of engineers. But it wasn't really until a couple years later that it started taking off.

But there was a really big change that happened internally I think. As we started shifting resources away from our primary interface, which was facebook.com, we started building up larger and larger teams that were working on our mobile apps. So, there were some interesting implications on web overtime, which we can get into at some point if you want. But I think that this actually just turned into a pretty big opportunity for us to innovate and create a lot of new technologies and new strategies for what I would call like Facebook engineering scale, delivering idiomatic iOS and Android apps, but with the size of our engineering organization in the scale and scope of our product.

[00:06:36] JM: My sense is that when the shift to mobile started to happen, there was a perception to Facebook building out mobile versions of the site was going to be a tax. But, over time, Facebook realized that it was an opportunity. Can you tell me more about the opportunities that Facebook found in that pivotal shift to mobile?

[00:07:07] TO: . Yeah, I think behavior really changes when you start talking about a device that's with you all the time versus a session where you get home and then you go on your laptop or your desktop computer and you use Facebook. So, we originally saw this as a tax primarily – And I don't know that I would call it really a tax, but we just saw it as a big shift and almost kind of like a really hard engineering problem and social problem, because we had the buildup the muscle. We had to build up the skillset for how to develop mobile apps. But the opportunity that presented itself later was, “Well, what can you do when somebody is using our services from anywhere and they have a device that's with them all the time?”

I think we started thinking about things like photo sharing differently, and video, and sort of real-time connection, and location sharing. Checking in became a popular thing around the time that we shifted to mobile. It was weird to sort of like check-in from your laptop. That really never took off. But when you're on mobile and you're out with some friends or you’re grabbing a bite to eat, it's really natural to add the location where you're at and tell a little story about that.

So, the opportunity that presented itself I think was really around the shift in behavior that results from having your device with you all the time versus this kind of like primarily like, “Let me sit down and check out a Facebook experience.”

[00:08:34] JM: I think Facebook has encountered some fundamental engineering challenges over its history that have been really formational in how the company developed its engineering practices. So, there were the difficulties of scaling a massive, highly interactive web application. Perhaps the most interactive web application up until that time, and it was a PHP application. There were also the difficulties of this shift to bringing that same application experience to mobile. Can you tell me about how the foundational engineering difficulties have characterized Facebook's engineering practices?

[00:09:20] TO: Yeah. So, we had talked about – I mentioned the sort of opportunity that we saw in the product it and you were talking about how the shift to mobile was a tax. So I think on the engineering, this shift presented a lot of you know foundational challenges, right?

So, we went from a world where basically everybody that was hired to work at Facebook as an engineer was hired as a generalist. We call them product generalists or systems generalists. But the idea is like you can kind of join Facebook and get ramped up on the software stack and then build kind of any products or any piece of frontend infrastructure or anything.

The foundational challenges that we had I think during the shift to mobile as a company was we needed to start hiring specialists more than I think we had ever done at that point. Because we wanted to ramp up quickly on having like pretty great feeling and pretty capable iOS and Android apps, we started hiring as many specialists as we can and also training existing engineers on those software stacks.

So, the result of this I think was a pretty transformational shift for Facebook engineering, which was everybody is kind of like treated the same almost and you go through the same onboarding process and you learn to use the same tools and everybody kind of builds features more or less the same way.

I've always been more frontend or user experience focused. So, I can't speak to some of the infrastructure challenges as well. But I would say on the sort of like product side or the user experience side, what ended up happening was we had to have a lot of duplicated effort. So, you went from a world where there was a single events team that built the events product to, for a period of time, there was the iOS team and the Android team and they independently were responsible for the events feature.

So after a while, you had deviations that were not intentional differentiation. It was just accidental or incidental differentiation. So, this was a foundational thing that we needed to sort of fix and we actually reoriented the company around this sort of product groups rather than what I would call functional groups or specialist groups.

© 2019 Software Engineering Daily

So that was a pretty big shift that happened. But at the same time, we started – I feel like as a company, engineering-wise, we just started slowing down. So we had a lot of challenges around how do we get engineers to be able to move as quickly as they used to. There was a massive foundational change when we had to go from just hit any endpoint and fetch whatever data that you need for whatever you're rendering to a world where everything is rendered on the client. So, we need to minimize the number of round trips over the network and we needed to know have this hierarchical data fetching strategy, and then we needed our clients to become more declarative over time. So, there was a ton of shifts that happened and I'd be happy to sort of deep dive on any one of them. But when the company shifted to mobile, engineering-wise, we almost became a different engineering organization.

[00:12:29] JM: One of the motivations for this as we were talking about before the show is to illustrate the differences between engineering at a place like Google and engineering at a place like Facebook. To illustrate how the products and the way that the products grow ended up driving the formation of the engineering organization, the development of the internal tooling. Then the later stages, the development of open source projects that end up being successful. Can you describe how the difficulties, the challenges of Facebook, those unique challenges ended up turning the organization into a uniquely good engineering organization that is differentiated from other orgs out there, perhaps like Google, which many regard as the gold standard of engineering organizations.

[00:13:30] TO: Yeah. There're probably quite a few things I could compare and contrast here, but I can just describe the sort of environment that led to a lot of the creation of a lot of the open source projects that were known for. One thing that we've always been very adamant about is that our open source strategy has never been about coming up with the one solution to rule them all. We're just a little bit more self-aware than that. I think when we came up with things like GraphQL and React and all these other technologies, they were in service of solving a specific problem for us.

So, I think that one thing that's really interesting about Facebook engineering is like every engineer has access to all of the source code. We have this mono repo – More realistically speaking, there're two repos, but all of our code is sort of in the same place. So, you can identify problems in the code base or in the products or in various different features, and you can try and

identify like repeated problems or repeated patterns and try and capture those things and then sort of make them go away so that the next engineer that comes along has to worry about fewer things.

I think something is interesting is when we do this at Facebook, we're operating inside of this mono repo. So, we kind of – You don't build something in isolation and then say, "Let me go find a client and see who will use it." You sort of build something in the place where you're trying to solve the problem. Then for open source, we extract. So, it's different than you build something on GitHub and you try to apply it to our code base. It's you build it in the code base and the place where you're trying to solve problems, and then you extract the open source software from there. So this I think is actually pretty unique of Facebook.

The other sort of side effect of this is that we don't do very much what I would call vendor. Vendoring to me is this idea that the maintainers of an open source project or of any dependency, really, operate on it sort of in isolation and then run releases of that thing at any product team that's using it or any downstream client of that thing at their own pace.

So what this means is that at Facebook, if you use React, you use of the one version of React that everyone else uses. So this has uniquely positioned us to have to care about backwards compatibility in a way that I think a lot of other open source projects don't need to. Everybody says, "Oh! I'd love to work on the React team." But it's pretty interesting, a lot of what the React team spends their time doing is just upgrading Facebook's code base for everybody else who is using React.

So, we're always using it nightly. We're always using the absolute latest version of all of our open source projects. I think as a result of that, our thinking around open source is we don't open source any software that we don't use our self.

So, there's interesting characteristics at Facebook engineering early on that led to this development, but I think it's actually served us really, really well, because you see a bit less of these sort of like competing strategies, and this team doesn't know what that team is doing, and competition for clients and things like that.

[00:16:46] JM: There's a role at many technology companies called platform engineering where you are building technologies for the internal users of the company to make their software better, and it sounds like React is an open source version of a platform engineering team, because you're building this platform that serves the people within Facebook and it just happens to be open source.

[00:17:22] TO: Yeah, that's kind of right. I think one of the things I've always guided the React core team to focus on is first identify the problems that are just common to anyone using React. So, a lot of folks try to cite this tension between, “Do you focus on internal priorities, or do you focus on external priorities?”

The way that I’ve been thinking about this is the fact that React is open source just kind of gives us more data points. But we have a number of problems that both internal and external users of React face. Then we have obviously a handful of Facebook specific problems. Then, obviously, the community as a handful of community-specific problems that aren’t relevant to Facebook because we have some custom infrastructure or whatever. The team, I think the Real core team, has focused all of their time on what I would call that first bucket, which is the problems that are unique or that are relevant to all React users, whether they’re internal or external.

So, yeah, the open source teams get to a point after they’ve sort of extracted that open-source piece of technology from that mono repo I mentioned. They get to this point where they’re actually kind of treating that thing as – Or treating all users of that thing as equal and trying to just improve it for everyone. There are times on the React core team will jump in on a Facebook specific issue and there are teams – There are times when the team will jump in on community-specific issues. But, in general, we never run out of interesting problems in research and the sort of overlapping space.

So, yeah, they are kind of like a production engineering team in that regard, but I call them like a bit of a hybrid. There's research happening. There's iteration. There's upgrading of our code bases. There's a lot of collecting feedback and making sure we’re collaborating closely with the most important partners both internally and externally. So, it’s real small but highly leveraged team.

[SPONSOR MESSAGE]

[00:19:31] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[00:21:23] JM: When you work at a place like Facebook, or Google, or Netflix, you often see problems that the rest of the industry are likely going to encounter in the next 5 to 10 years, and you see them right now. Are there any particular problems that you're seeing today in infrastructure that you're dealing with firsthand on one of the teams that you manage that you think is a leading indicator of some engineering challenges that the rest of the industry is going to face in the near future?

[00:22:03] TO: Yeah, this is a great question. I think if I can give you one that I saw early on in the days when we first started React that I think is actually still relevant today. One of the things we saw was that scale is not just about the size of your code base or the scope of your product. Scale is also about effectively adding engineers to a product.

I think one of the things we saw early on that large companies are almost certainly a leading indicator of is as you add more engineers, things get harder and harder to maintain and iterate on. So, one thing that I've noticed is an industry-wide shift now towards strategies that are much more declarative, much more functional and much more immutable and also this rise in concurrence in asynchronous programing, because declarative code is easier to – If you have truly declarative functional code, imagine all of your functions are pure and nothing does mutations anywhere. You have a lot more flexibility and predictability on that code.

So it makes it not only easier to maintain and it makes your code base more reliable over time, but more importantly, it makes it so that new engineers can get on-boarded more quickly with confidence, because there's a set of guarantees. We're moving away from this world of imperative mutation and let me grab that thing and mess with a little bit. Then data binding was a common pattern. I think it still is in many cases.

Yeah, so I would say we're moving from an imperative, maybe even object-oriented synchronous execution immutable data structures world into a world of more functional immutable declarative paradigm so that we can have better asynchronous programming, better optimize for taking advantage of multiple cores on devices as our applications get more and more sophisticated. Also so that we can grow our engineering teams to a larger size while still maintaining high-performance, high-quality and things like that.

[00:24:12] JM: I'd agree with that, and I think there has been less and less use of the term callback hell since I started the podcast. Is it consistent with your internal Tom Occhino verbal trends ticker that's going in through your head?

[00:24:31] TO: Absolutely. I think with the introduction of async/await into JavaScript, at least for most folks, callback hell was kind of like that was it after that. Then with the introduction of React, I think we have these – I don't know, just well-defined seams in our application where

you're not really thinking about that crazy nesting at least in the UI layer that you used to have to. I think with things like the introduction of hooks inside of the React core, for example, at least in our user interfaces, we're talking about less and less nesting over time.

Yeah, I don't hear anyone talking about callback hell anymore. So maybe this is maybe Facebook and larger company engineering as a leading indicator that there is light at the end of the tunnel here for anybody who's stuck in callback hell still.

[00:25:18] JM: What's the 10-year vision for React?

[00:25:21] TO: Yeah, this is a great question and something that I've been thinking a lot about lately. So, look, we stumbled upon something really interesting and really important here, which is that when you don't have to think about mutations so much and when you don't have to think quite so much about the intricacies of the underlying platforms that you're running on, engineers feel empowered to focus on the most important problem, which is like actually the thing that they're trying to build rather than getting caught in the weeds of sort of what I would call like the worst parts about software engineering.

I think the 10-year vision over-time, the idea is like once you learn how to use this stack and hopefully get simpler and simpler to learn over time. I can talk more about. But once you learn the stack, you sort of as an engineer are empowered to at least have a starting point on sort of any product or any platform that you want to build for and we're a really long way away from this. But I think React Native and React360 are a step in the right direction.

But I imagine that future for React, which is there is a very, very low barrier to entry for getting started with something. Then, over time, you can you know build something that is more and more expressive and more and more sophisticated, but you learn about the complexities of the underlying platforms or you learn about the complexities of computer science itself progressively overtime rather than being frontloaded with all of the knowledge and history of computer science and electrical engineering from day one.

One of my motivations for still working on React after so long and even still being at Facebook after 10 years is I actually really do feel like we are in many ways lowering the barrier to entry for

computer science. I went to school for computer engineering in the first day. Lecture hall was just packed, people standing in the aisles. By the end of the semester, I would say roughly 2/3 of the class had just dropped computer science as major, because the barrier to entry was so high. It just didn't relate to real life.

But now, in a more app-centric user experience-centric world, people can understand how looking at a React application relates to what they see on their screen. So, the 10-year vision I think here is just enabling every single engineer who sort of learns this pattern, learns this paradigm to be able to be more leveraged and be more capable and build for any platform, at least as a starting point. We'll never get rid of the underlying strengths of each platform. We're not trying to hide or paper over those things either, but what we want to do is make it so that you don't need to learn in detail about those strengths until the appropriate time, what I call progressive disclosure of complexity over time.

[00:28:12] JM: My sense is that when React got started, there was no vision that it would become a kind of project where you would be having to solve low-level algorithmic challenges. But over time, it has become this piece of infrastructure where the critical path to React rendering and executing is very performant, and tweaking has become quite an art in improving the performance. Can you tell me about some of the difficult engineering problems that you've had to solve in making React performant?

[00:29:02] TO: Yeah. I mean, as I like to joke, there's some real computer science happening here. But, yeah. I mean, one of the biggest problems we've had to solve is how to make product code, which can literally be anything. Somebody can have a wild true. How do you make product code sort of performant or good enough by default so that you don't get bogged down in premature optimization and things like this?

So we've tried to guide people who use React towards patterns that we think are sort of just good enough by default, and then you can kind of – You have a bunch of vectors to explore to make it even better. But that I would say was one of the hardest challenges. We would profile these apps early on in the React days and we would find that like, "Yeah, we can make React you 10% or even 20% faster," but that's not going to make your thing that took five seconds to render take an appropriate amount of time. We needed order of magnitude or step function

improvements in some of this product code. So, what we did was we captured a bunch of repeated patterns and we just tried to make them go away.

The move towards functional components I think is important, because it gives us even further levers that we can explore in the future around sort of optimizing compilation and static analysis and dead code elimination and static extraction of repeated UI and things like that. But there've been a lot of challenges over the years just making sure that this – You look at this at a high-level and you think about what React is doing and like there's no way intuitively or just like there's no way this is going to be fast enough. You're going to compare at everything. I used to rendered everything I'm about to use to render. Then you're going to like update only the piece – This is going to be so much bookkeeping. It's going to be so much diffing.

But in reality what we've done is we've taken a very hard problem and we turned it into something that machines are really good at, which is basically math, its comparators and nested for loops. Actually, they're very fast. So the original implementation of React I think had this N-squared algorithms for doing the virtual dom diffing or whatever we called it back in the day. We were eventually able to get that to a place where we have like linear time.

So, it's very rare these days that React itself is the bottleneck. It's actually the amount of code that you download and what you do in your product code that might not be conducive to sort of performant execution even outside of React and things like that. Yeah, I think over the past three years, especially ever since our, what we called the React Fiber rewrite, this went from being an optimization on a very simple to explain rendering strategy to being a proper sort of computer science research project with practical application and backwards compatibility.

So, I'm even more excited about some of the optimizations and new strategies we're working on in the future, especially around React Native and how we're thinking about better interoperability with underlying native platforms. The long story short of this is in order to make React Native have the ability to render synchronously, we had to make React the entirety of the framework asynchronous so that we could interrupt rendering at any time so that we could have synchronous execution on a main thread of a mobile device, for example.

So, yeah, there're some really interesting things happening here. There're been a lot of challenges around performance. One of the challenges we'll always faces is how much product code gets downloaded and the size of the framework I think specially on low-end devices and emerging markets. We're still sort of constrained by how long it takes to download parse and evaluate JavaScript bundles and things like that.

But there's a time of opportunity left. We're just starting to scratch the surface. The other thing I'd say is like we are just so much closer to the beginning of this than we are to the end of it or the winding down or the moving on. React has evolved a lot over the past five years, and the React code that people write today looks nothing like the React code that was written five years ago. But because this revolution, if you will, happened incrementally and evolutionary over time, and we've maintained backwards compatibility at every step of the way. I just have a lot of confidence that there's a ton of runway left.

[00:33:25] JM: With React Fiber as an example, React Fiber being this rewrite that you're talking about that required a lot of changes across the React code base, can you describe how you manage a big initiative in an open source project like this where you have a large number of engineers that are reporting to you? How do you plan out a strategy for managing those engineers and what's your process for managing that on an ongoing basis?

[00:34:02] TO: Yes. So, I think there's a sufficient amount of planning that has to happen upfront with a project like Fiber. I mean, it's easy to talk about now, in retrospect, but Fiber was pretty scary, because it was a complete ground-up rewrite of React. The goal was to make it so that on day one no one noticed, right? So, basically you can drop this thing in and it will just work with your existing application.

The team wasn't very large, but basically the way that we managed it was, first of all, fostering compatibility from the beginning and making sure that we have unit tests that capture all of the behaviors, even the ones that we think we wanted to change overtime and captured all of the nuances and idiosyncrasies of the old strategy, the old –. We call it stack was the old version of React, React Stack and React Fiber.

So, over time, we were simultaneously iterating on this new strategy, which was based on some research and also making sure that we're adding new unit tests that capture all of the nuances of the old implementation. Then we were perpetually just deploying this inside of Facebook at our scale.

So, if anything would break anywhere, even if it was terrible product code that we would love to have rewritten or even if it was our old code that was outdated, we would strive to make sure that we were compatible with it for this first iteration. Then overtime, we would start to capture those bad patterns and make them go away.

So, when React 16 came out, ideally, most folks upgraded to React 16. After they fixed a couple of warnings here and there, it was actually pretty seamless. But the great majority of the entire code base is completely new.

So, managing that team, I think it was – They were pretty excited about this beautiful foundational end state, but they also recognize that software is not about a destination. Fiber, even though we had sort of a milestone of shipping that in site, we recognized that this is a journey and that was just one milestone on a longer journey.

So, yeah, I mean, the team was pretty small. It wasn't really even about managing a large initiative at that point. It was about making sure that everybody who depends on us can continue to depend on us. Again, we don't have vendoring. So, when we wanted to start making this work internally, we were building up all of the necessary infra to make it so that we could switch between these two versions seamlessly and we could upgrade one part of the code base at a time and that we were working with all of the clients internally of React to make sure that their code continued to work no matter how non-idiomatic some of it may have been.

[00:36:49] JM: What's the biggest mistake you've made as a manager?

[00:36:52] TO: Not trusting enough the people that I support early on. So, it took me a long time after I transitioned into management. This is probably about six or seven years ago now. But after I transitioned into management, I fell into this trap that I think a lot of new managers fall

into, which is you still are responsible for everything. There's just kind of more stuff. So you need to make sure that like everything is going okay.

Actually, successful management has nothing to do with that. Successful management is about providing support and encouragement and constraints and expectations, and then enabling the engineers that you support to do it their way, not your way. So, I got feedback for probably my first two or three cycles that I was kind of like micromanaging and I had to be involved in every technical discussion. I don't think that any of these was the outcome of this was detrimental. It was certainly just a learning experience. I think that I grew as a result of it. But, oh! I'm sure I've made more mistakes than that over the years. But that's kind of one thing that stands out, because now that I manage managers and managers of managers, it is one of the things that as we transitioned and developed new managers, that comes up quickly and that I like to always make sure everybody learns.

I've also developed a number of principles based on things that I've learned. One of the principles I have is like people hate surprises. You may say that a surprise party is the exception to this rule. But, actually, the only surprise parties that go well are the ones that the guest of honor actually knows about secretly. People don't even hate bad news. They can handle bad news. They just hate to be surprised with. So, I'm sure I've made a bunch of mistakes around surprising people with information.

I now operate incredibly transparently with everybody that I support so that they have as much information as possible to being able to make decisions. I'm sure I could pull up my list of other principles and then we could talk about some of those if you're interested as well.

[SPONSOR MESSAGE]

[00:39:11] JM: As a software engineer, chances are you've crossed paths with MongoDB at some point. Whether you're building an app for millions of users or just figuring out a side business. As the most popular non-relational database, MongoDB is intuitive and incredibly easy for development teams to use.

Now, with MongoDB Atlas, you can take advantage of MongoDBs flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security, and high-availability, and data recovery, and monitoring, and elastic scaling.

Try MongoDB Atlas today for free by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas as well as support Software Engineering Daily by checking out the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se.

Thank you to MongoDB for being a sponsor.

[INTERVIEW CONTINUED]

[00:40:34] JM: The earlier days of Facebook when the organization was smaller, my sense is that it was easier for people to move about the organization as they pleased and perhaps start new projects in a decentralized fashion. I'm just saying that as a function of the size of the company, changing over time. So, it in in interviews with Nick Schrock and Pete Hunt, you really get the sense that there is just this lateral mobility that people have and they can move fluidly around the organization and it's kind of a flat structure. But, overtime, that's very, very hard to scale. I think you alluded to that when you saying you're becoming a manager of managers and there're layers of management, and that's just typical. It's very hard to maintain that completely flat organization. How has Facebook adjusted as an organization in moving from those early days where you have this really enjoyable element of a culture where people can move around so fluidly towards the more mature management hierarchies that become necessary?

[00:41:51] TO: Well, the first thing I'd say here is we are very, very much preserved. This fascinating phenomenon, I'd say, of engineers being very, very mobile and being able to move. We literally call it engineering mobility, but being able to move between teams.

Now, there're some guidelines here. You can't join the team for a month and then drop a project on the floor and then pick up a new team. But we actually encourage engineer still after they've

been on the same team for a year to explore what's called a hack-a-month and try something new out.

The reason that Facebook has been uniquely positioned to be able to foster this, and I actually think this engineering mobility has been a strategic advantage for us as an engineering organization. The reason we've been able to do this is because of a program we have called boot camp.

Typically, for an engineering manager, and I'll talk about manager mobility in a minbute. But for an engineering manager, when somebody decides they want to leave your team, it's almost like alarms are going off. You're freaking out, because it's really hard to hire somebody new, especially in Silicon Valley and especially in the tech industry. I think finding good people takes a long time and the entire end-to-end recruiting process can take months.

But because of the way that we onboard new engineers, there's a tool that's called the jobs to all internally, and at any time, I can go and look or any engineer can go and look at all of the positions that are open. We call these positions, or PID's, position IDs. But you can think of them kind of like desks, right?

So, there's a team that has two desks open and there's a team over here that has one desk open. At any time, if I am doing well or if I'm just ready for something new, I can kind of talk to my manager about occupying one of those other positions. It can start out as an informal, what we call hack-a-month, which is like, "Let's try it out and see how you like it. If you don't like that team, you can go back to your old team or you can try something else."

But we very much continue to have this work at our scale, because there is an endless supply seemingly of new engineers. So what happens is if someone tells me that they want to try something new, or if I think that they should try something new, either because there's a new company priority or because I think I found something that really aligns with their skillset that could really use their help. They can switch to that seem. I have the open position now. Their position doesn't go with them. So I have an open position, and then I can go back to boot camp and hire somebody else to sort of backfill the position that they had. I can do this in a week or less.

So, now, I have the opportunity to grow somebody else on my team. Let's say a tech lead leaves my team. I can grow the next person on my team into a tech lead position. They have a new opportunity. I can onboard the new person on to my team. They have a new opportunity. They're learning and exploring this environment. The person who left my team who was a tech lead now has a new opportunity for growth and impact on a completely different team.

Simultaneously, for Facebook engineering, they've taken all of the knowledge and information they have and the relationships they have with them. So, we have this phenomenon of this very interconnected sort of engineering organization.

Now, for managers, as you alluded to, it's actually a little bit harder because now I have this support structure where a lot of different people are depending on me. But we even have programs for all the way up to director level and we've seen this even in the executive level for people to be able to – It takes a little bit more lead time and a little bit more planning, but to be able to find something new. Because we foster this, what we see is that, at least anecdotally, we have a little bit less attrition, because when you get sick of what you're doing, the only recourse is not leave the company and go somewhere else. It's explore a different part of the company, which may have a very different product or a very different technology stack or a very different set of teams and practices and things that you want to try out.

So, we've seen this. Recently, we had a director leave one group working on mobile and go work on VR and we had a manager leave a team working on some of our youth infrastructure to come work on some of our web infrastructure, or web products. So, we actually like – This kind of unprecedented. There's no real – Engineering managers don't see it as a real downside when somebody decides they want to do something new. It's kind of like, "Oh! That's great. How can I enable you to work on something that aligns with your strengths and your passions and delivers a lot of value to our community and to our company and things like that?"

[00:46:25] JM: There was a series of blog posts from Airbnb about their migration away from React Native. What was your reaction to that whole affair?

[00:46:37] TO: Yeah. This is really interesting. I think they had shared those blog posts with us very early on, and I think the early take away was like, "Wow! There's no way for this to look

good for React Native." Actually, I think the truth of it is that the technologies that we built all with them, we have always said, "This is the thing that works for us. If it works for you, great. It's open source and we'd love to collaborate with you and identify opportunities where there's overlap or where we can work together. But if it doesn't work for you for any reason, no hard feelings at all."

We still have a great relationship with Airbnb, especially some of the folks that used to work on React Native at Airbnb, and I don't think there're really any hard feelings here. I do think that not every technology is perfect for every situation, and we've never – Again, I said at the beginning, we've never tried to create – None of our open source projects have ever tried to be the only player in a space or the last solution to rule them all. We've been working really closely with Microsoft and a bunch of other companies that are very heavily leaning into React Native and getting a lot of value out of it. We ourselves are getting a lot of value out of it.

So, our partnerships shift overtime. But I think there was no hard feelings there. I think there's a ton of strong engineers at Airbnb, and while we were working together, we got a tremendous amount of value from working with them.

[00:48:00] JM: One of the differences between Facebook and some of the hyper-scale engineering companies that have come after Facebook is that cloud providers are now available, public cloud providers. What's your sense of how Facebook engineering differs from a company that is built on the cloud?

[00:48:24] TO: Yeah. This is interesting. I think the entirety of our stock, from the hardware to the pixels on your screen is sort of vertically integrated at Facebook. A lot for historical reasons, but we still take advantage of a ton of software and open source software and we contribute a ton back and we have open source hardware initiatives, the Open Compute Project and things like that. But I think the big difference is that as these cloud providers have become bigger and bigger, played a larger and larger role in software development, the companies that depend on them can actually focus on fewer things. You don't need quite as large of an infrastructure org when you're not literally building your own servers.

So, I think what this has enabled is a lot of innovation that wouldn't have otherwise happened if you had to build an entire team from end-to-end. So that's what I think software is sort of moving so much quicker and the progress that we're making is happening a lot faster. I do think that anytime any of those companies get to a sufficiently large scale, they will want to have more control over not just the infrastructure and the technology stack itself, but also sort of their own bottom line.

So, if there's a way for you to manage your own data center and things like that, it's a good problem to earn. But I think that the great thing about all of these cloud providers is that they have enabled so much innovation that probably otherwise could have been stifled by the fact that it's just too hard to spin up an infrastructure team to deliver all of the services that these cloud infrastructure providers are providing.

[00:50:04] JM: You manage both the React team and the web core team. What are the differences between managing an open source initiative versus managing close source internal infrastructure?

[00:50:17] TO: Yeah, this is a good question. I think one of the things we factor into the roadmaps and the schedules for the React group, which includes React Core, React Native and Relay, is that there will be some amount of time that every engineer is spending on GitHub, or engaging with the community, or outreach, or support, or whatever.

So, I think one of the differences is that almost all of the meetings of the web core team or the partnerships, all of those things, they're actually all internal and they're all kind of managed with our internal tooling and all these other things. But the React core team, for example, has a weekly meeting with the Chrome team. The React Native team recently went up to Washington to meet with the Microsoft teams and working on React Native. So, there's a lot more sort of interoperability that we've kind of factor into our planning and our scheduling.

But other than that, I mean, the really nice thing here is that these teams work super closely together. So, there's a bunch of projects right now that the React core team is working with the web core team on we use Relay a lot internally. So, the web core team is sort of the first clients and most important clients perhaps of Relay. So they get to work really closely together. So

there's a ton of overlap. I like to draw this diagram of my entire org where every single team has an adjacency and has a collaboration vector with every other team.

So, yeah, I think the biggest difference is just that we factor in this idea that there's going to be a little bit of community engagement and outreach and managing GitHub and poll requests and issues into our planning such that the amount of projects that you can take on without that is probably slightly higher than if you do want to maintain a healthy community. But that's pretty much the only difference I would say.

[00:52:11] JM: A quote from an engineer who worked with you, Tom is good at making a group at a big company feel like a small company. Why is that important?

[00:52:23] TO: Well, I think it's important because I feel like people do their best work and enjoy their work the most when they don't feel like they're just another cog in the machine. One of the things that's really important to me is that everybody in my group has someone else in my group that they are friends with and close with.

I think one of the biggest indicators of happiness in a career is the answer the question, “Do you have a best friend at work?” I think that it's really important to operate in this way that is like independent of, but complementary to the larger system, the larger company, or ecosystem, because you have to be insulated from the swings. Those are both good and bad. But you have believe in what you're doing and you have to understand how it connects. But you also just have to like coming to work. You have to enjoy your job. You have to be optimistic. You have to be in a good mood. People don't do their best work when they feel isolated or they feel not connected to their team or they feel like they're all alone and whatever they’re building, or maybe something is weighing on them from outside of the group.

So, I know all the names of everybody in my group I meet with literally every – I think it's somewhere around 60 people in my group now and I meet with all of them. Everybody knew who joins, and I have lunch with the team and we, sometimes on Fridays, if people don’t have anything going on, we’ll play Nintendo Switch together in the office and things like this. So, I want people to really enjoy their work. I want them to think about going to bed on Sunday and

actually looking forward to get into the office on Monday morning. I think we've done a pretty decent job of that.

[00:54:15] JM: What's something that you believe about management that other managers disagree with you on?

[00:54:21] TO: I am a very encouraging and optimism, and positive reinforcement-driven manager. I believe in the carrots not stick philosophy. I have worked with managers who are sometimes very effective, but who basically layouts requirements or demands. Then if people don't meet those requirements or demands, that person is the problem and we need to replace them or find somebody new.

But my management philosophy has always been that everybody has some unique strength that they bring to the table and all it is is my job is just to identify it and to amplify it. I think something that's controversial is I believe that if you identify a weakness of an engineer, you actually just have to do more work to take that weakness off the table.

What I mean by that is if you have an engineer who really loves like algorithmic complexity or really loves you know fighting fires, but maybe they're really bad at communicating about their work or justifying their work. It is not your job as a manager to force them to communicate about their work or to force them to justify their work better. It is your job to make it so they don't have to. So pair them up with somebody who can communicate about what they're doing, that they get along with and understands them, and you do the work to justify it and connect it to something bigger.

So, yeah, I really try to foster an environment where people can focus on what they're good at, what they're passionate about. As managers, we'll do a little bit of extra work if necessary to connect it to something bigger. Obviously, we need to course correct once in a while, but we attract engineers who are kind of already bought into kind of what we're trying to do and nobody really takes advantages or takes for granted that situation.

Yeah, I'm very much the sort of optimist, the positive reinforce. I reach out to my engineers and my group when they do something well and I recognize them for it. If something bad happens or

somebody makes a mistake, we learn from it and move on and you provide encouragement, not condemnation, if you will.

[00:56:42] JM: What you're saying is that you want to cater to an engineer's strengths rather than coping with their weaknesses. This is a great ideal to strive for, but it can lead to a kind of stable matching problem where the number of tasks that you have available or the different tasks that are available, they may not copacetically with the engineers who are available, who, “Oh, I want to work on something with algorithmic complexity,’ and you say to them, “Well, sorry. We really only got this really boring thing that's like writing unit tests to test the frontend. Sorry. That's all we've got right now.” Is it ever a struggle to be able to solve that matching problem and how do you cope with that?

[00:57:33] TO: It hasn’t been a struggle thus far, and I think the reason for that is because if you create an environment where the people you support know that you have their back, they will have yours. The people that I support recognize that if I come to them with a crisis or a situation where this really needs to be staffed or we really need to work on that, they will find ways to make sure that that work gets done even if it's not their number one choice.

But, you're right. It is a stable matching problem and it's in constant evolution. There's never a situation where everybody is just perfectly happy with everything that they're working on and the team is having the most amount of impact possible. But what I would like to do is optimize for the amount of time that engineers get to spend doing the work that they love or the work that they are very good at.

If an answer to that question, if you ask an engineer that question and they can say, “Oh! I spent about 70%, 75% of the time that I work, doing work that I really enjoy or work that I'm good at or work that I love,” then the other 25% of the time, when they need to be doing that work that just needs to get done, they have a higher tolerance for that because they know that in large part they get to focus on mostly the things that they’re passionate about.

[00:58:46] JM: Facebook is known for being willing to go against “best practices” of computer science or software engineering. What's the most memorable time when this has occurred for you?

[00:58:57] TO: Yeah, this is absolutely the release of React. But there've been dozens of times since, I'm sure. But when we originally open source React, I think our messaging was probably missed the mark, our initial messaging, and we focused on some things that probably were not the most salient points. But, this idea that you would embed your HTML in your JavaScript and that would somehow be better for maintenance of your code and reliability and extensibility of your code, the challenging established “best practices” literally came from that presentation.

There was somebody in the audience who had tweeted at us, “Look at Facebook rethinking established best practices as a joke,” and everybody kind of laughed, but we were like, “Hey, wait. That actually is what we’re doing. We have tried this. It has worked for us, and we are sharing it with you even though it's not the current established best practice.”

So, yeah, the release of React was certainly the first time that I experienced this at such scale. But Pete Hunt actually embraced this, and later that same year, gave a talk at JSConf EU, called Facebook rethinking established best practices. Yeah, that was the most obvious case, but I'm sure there have been countless times when we've done this again over the years.

[01:00:17] JM: Tom Occhino, thank you for coming on Software Engineering Daily . It's been really fun talking to you.

[01:00:21] TO: Jeff, thank you so much for having me. I had a really great time.

[END OF INTERVIEW]

[01:00:27] JM: Commercial open source software businesses build their business model an open source software project. Software businesses built around open source software operate differently than those built around proprietary software.

The Open Core Summit is a conference before commercial open source software. If you are building a business around open source software, check out the Open Core Summit, September 19th and 20th at the Palace of Fine Arts in San Francisco. Go to opencoresummit.com to register.

At Open Core Summit, we'll discuss the engineering, business strategy and investment landscape of commercial open source software businesses. Speakers will include people from HashiCorp, GitLab, Confluent, MongoDB and Docker. I will be emceeing the event, and I'm hoping to do some on-stage podcast-style dialogues.

I am excited about the Open Core Summit, because open source software is the future. Most businesses don't gain that much by having their software be proprietary. As it becomes easier to build secure software, there will be even fewer reasons not to open source your code.

I love commercial open source businesses because there are so many interesting technical problems. You got governance issues. You got a strange business model. I am looking forward to exploring these curiosities at the Open Core Summit, and I hope to see you there. If you want to attend, check out opencoresummit.com. The conference is September 19th and 20th in San Francisco.

Open source is changing the world of software and it's changing the world that we live in. Check out the Open Core Summit by going to opencoresummit.com.

[END]