EPISODE 330

[INTRODUCTION]

[SPONSOR MESSAGE]

[0:00:13.5] JM: For years, when I started building a new app, I would use MongoDB. Now, I use MongoDB Atlas. MongoDB Atlas is the easiest way to use MongoDB in the cloud. It's never been easier to hit the ground running.

MongoDB Atlas is the only database as a service from the engineers who built MongoDB. The dashboard is simple and intuitive, but it provides all the functionality that you need. The customer service is staffed by people who can respond to your technical questions about Mongo.

With continuous back-up, VPC peering, monitoring, and security features, MongoDB Atlas gives you everything you need from MongoDB in an easy-to-use service. You could forget about needing to patch your Mongo instances and keep it up-to-date, because Atlas automatically updates its version.

Check you mongodb.com/sedaily to get started with MongoDB Atlas and get $10 credit for free. Even if you're already running MongoDB in the cloud, Atlas makes migrating your deployment from another cloud service provider trivial with its live import feature.

Get started with a free three-node replica set, no credit cards required. As an inclusive offer for Software Engineering Daily listeners, use code "sedaily" for $10 credit when you're ready to scale up. Go to mongodb.com/sedaily to check it out. Thanks to MongoDB for being a repeat sponsor of Software Engineering Daily. It means a whole lot to us.

[INTERVIEW]

[0:02:12.3] Q: I'm really excited to have Tom Occhino on the show today. He manages the React team at Facebook and he's very, very involved in the open-source involvement within

Facebook. We're going to be focusing on talking about how open-source works within the Facebook organization. Before getting started, I love to just have Tom jump in and give a quick introduction of himself.

[0:02:35.4] TO: Hi. Yeah, absolutely. Thanks for having me on the show. I'm excited to chat with you. Just a little bit about myself. I'm been here at Facebook for a while now, probably about a little over eight years actually. I've always been involved in our frontend technology stack, and that started out with HTML, CSS, and JavaScript and a little bit of PHP, but eventually moved on from building products to building, what we call, product infrastructure.

Product infrastructure is the systems and frameworks and libraries and things that we build that power the products that we build. Part of that is React, so I got involved in React before it had a name, when this guy, Jordan Walke, created it. I've been managing that team ever since and heavily involved in some other things as well.

[0:03:20.5] Q: Going back to the very early days, I know Facebook started open-source efforts very early in the company's life cycle. There was the Cassandra database, the Tornado Web Server, and a bunch of other projects. Open-source was important by then and it's becoming even more important today as more and more of the software that we use today is becoming open-source.

As a role of open-source increases, I think people are starting to have this expectation for large organizations like Facebook and Microsoft or IBM to play a role in the open-source software that the community uses. I'm curious. What motivated Facebook to be one of the early movers in their efforts in contributing to open-source and being someone involved there?

[0:04:07.6] TO: I think a primary motivation was the fact that we were such a large consumer of open-source software. Even before I joined Facebook, the stack that we were building Facebook on top of was basically your traditional LAMP stack, and all of the technology is — PHP was a huge thing that we use. PHP and all the other technologies that we — MySQL and all these other things were all open-sources. We were motivated to give back, I guess. We're taking advantage of all of the stuff that exist in the community in trying to make it better

ourselves. If we come up with new things, when it makes sense, we'll try and contribute those things to open-source as well.

Some of our early efforts like Cassandra, and even a little later on, Tornado, after the FriendFeed acquisition, I think; there was a lot of server-side and low-level backend infrastructure focus for some of our open-source efforts. Then, I would say back in about 2013, we refocused on the frontend or started focusing on the frontend a lot more with especially the advent of React.

I think the answer to your question, why do we do this, or why do we feel compelled to open-source technology. I think it's because we were such heavy users of open-source. We always knew we wanted to be able to give back to the open-source community.

[0:05:29.5] Q: Interesting. That's super interesting. I've seen a lot of enterprise companies jump into open-source, and it just makes sense there. I guess for a consumer-facing social media company, what are some of the benefits or things you get from being part of the open-source community?

[0:05:50.3] TO: Yeah, there're a number of them. I think anytime we open-source software, it forces us to give it an extra round of almost vigor and try and make sure that it's in good shape. We make sure that documentation is in order, and things work, and it's sufficiently decoupled from the rest of our infrastructure and things like that. One benefit is it just makes our software better, we think.

Another befit is, obviously, on recruiting and brand recognition, a lot of people who have joined Facebook in the past three or four years cite one of the reasons they're really interested in joining is because they want to work at a place where something like React can be created. It's not even necessarily just about React or GraphQL or some of our other open-source technologies. It's about wanting to work in an environment where you're allowed to share what you work on with the world. That's been a huge thing for us.

I think we've also received tons of amazing contributions that have made the software better from external contributors, and it's acted as a wedge, almost, for us to be able to collaborate

and communicate more with other companies and in completely different industries or even in the same industry. The benefits of openness and communication are core to our mission as a company. Open-source enables us to carry that out on a software front, the technology front.

[0:07:13.9] Q: Yeah, I definitely want to come back to that point about open-source being attracted to hiring new developers. I think that's a very interesting and important point. Kind of taking a step back, you mentioned how Facebook started open-source mostly in a lot of the backend database technologies and then has more recently, moved into the frontend. Can you give listeners a lay of the land of all the different areas of Facebook has open-source technologies today?

[0:07:42.6] TO: Oh, gosh. Yeah. Everything, basically. On the software side of things, especially on the frontend, we have frameworks for building UIs. We have frameworks for doing data fetching, big to small. Things as large as React in GraphQL and also things as small as — Recently, we open-sourced a single function called the existential function, IDX, which is a very, very small library that serves a single purpose and serves it well. That's on the frontend JavaScript and side like that.

We also have tons of server-side infrastructure around everything from database systems, and key value stores to caching systems, and build systems, and editors, and all sorts of different software on that side of things as well. We also have projects like our Open Compute Project, which is all about open-sourcing the specifications and designs for the hardware that we use to power all of our services.

In our data centers, we have racks and racks of computers and servers, and the designs for those servers are actually open-sources. It's part of the Open Compute Project. It's everything from hardware to software and everything in between. It's the variety of things that we open-source.

[0:09:02.8] Q: Jumping into the team and how that comes together, I know that you're mostly focused, or overstay in the React team. I'm curious, when you're making a decision to open-source a technology, how do you find the right people to kick-start the effort?

[0:09:22.7] TO: An interesting thing about the way that we write software at Facebook is the people that author the code or the people that support the code, that doesn't mean just through open-source and things like that, it means when your code goes out to production, if you built a user-facing product or you built a library or something like that, when your code goes out to production, you are the QA. As an engineer, you're on the hook for making sure that your software works.

The process of open-source, something is very similar. It's not like the authors author it, and then another team takes over and starts to maintain the open-source project. The key driver for whether or not it makes sense to open-source something is — One of the key drivers is are you committed to maintaining it as an engineer? If Facebook determines that it's valuable to open-source, and we're using it in production, and it's working for us; those are the prerequisites.

The next question is: are you going to maintain it? Are you going to be contributing to the project? Not necessarily indefinitely, but fostering a community in building up a first line of defense in things like this and getting other people helping and supporting, because what we don't do — With some projects very early on in Facebook's life, we did this incorrectly. We would throw things over the wall. You have something and you don't necessarily even open-source it, you just make the source code available, and people can do with it what they want. That's not the way we treat open-source anymore.

I think the question was; how do you decide if something is good for open-source? There's many other, actually, facets that go into figuring out if we should open-source something, but the big driver is; is the engineer motivated to build a community and maintain the project and be a good steward of the project.

[0:11:16.5] Q: Interesting. It falls upon the engineer themselves?

[0:11:20.5] TO: Exactly. And the team, yeah.

[0:11:22.7] Q: Yeah. Then, if the engineer is not that interested — Because building an open-source tool and then maintaining it are very separate things and different skill sets, too,

altogether. If an engineer is not interested in maintaining it, but they want to open-source it, is it possible for that to still be open-sourced?

[0:11:44.5] TO: It's definitely possible. They basically just have to find people who are motivated to be — Not necessarily just maintainers, but be the folks that are going to drive for even a small community to be created or set up an RFC process. There has to be somebody who is committed to the project or else, we wouldn't open-source it, basically.

It doesn't need to be the engineer themselves. They just are responsible for finding the engineers that are willing to put in that efforts and make sure that we're not just throwing something over the wall. Sometimes, I imagine, we probably have things that we would consider open-sourcing that are actually just reference implementations.

We would say, "Hey, this isn't actually a thing that we're going to maintain, but it's a problem that we solved. Maybe our solution is to couple to our infrastructure, or maybe we don't really want to maintain it as an open-source project. We want to continue iterating, but we want to share our solution with the world." That would be a different type of release if we were to do that. I can't think of any examples where we've done that before, but I could just imagine us saying, "Oh, you know. Let's not open-source it, but if people are asking for it, let's still make the code available."

[0:12:55.6] Q: Wasn't GraphQL one of these situations?

[0:12:58.4] TO: GraphQL interestingly started out as — It had no intention of being open-source. It was a very specific problem to a set of problems that we were having with how we do data fetching in our apps. I think once we started deploying it to React applications, it became a natural candidate for open-source because if people are using it successfully alongside React at Facebook, that might be an indicator that it's relevant elsewhere. Engineers were generally very happy with using GraphQL, plus React.

At the first React Conf in 2000 — What was it? 2015. When we originally released and announced and talked about GraphQL, I think we were a little surprised and overwhelmed by

the reception. It was amazing to see it, because we had been using it internally for a long time and we knew it was working, but we didn't know that, necessarily, the problems would resonate with everyone. That's one of those cases where it wasn't originally designed to be open-source. It wasn't originally built to be open-sourced, but it was a natural fed. It was a good candidate for it after the fact.

Other projects are designed with open-source in mind. From the beginning, maybe they'll start out with their source code on GitHub, in a private repository, and our intention will always be to eventually open-source this.

I think one project that comes to mind is called Fresco; an image library for android that makes image rendering a lot faster and more performant and more efficient. From the beginning, we were like, "Okay, if we are solving this problem, we have talked to the rest of the community. We know a lot of other people are facing the same problems. Let's go ahead and just set this up so that if it is successful here, it'll be very easy to flip a switch and make it public."

We have both. We have projects that are after the fact, we go back. React was one of these as well. React wasn't originally designed to be open-source as well, but when Instagram wanted to start using it, and we needed to decouple from out Facebook-specific infrastructure, it became a good candidate for open-source. We'd already done the work to decouple it, and it was a general purpose thing. Then, other projects, the other cases, from the beginning, you know that if this works, you want it to be open-sourced.

I would say that case is becoming more common these days, especially as we hire more and more people who are passionate about open-source. Everybody that we work with in the open-source community, they all assume that everybody is really excited about open-source and passionate about it, but it's actually not that. There're actually quite a few people that appreciate and feel they benefit from their code not being open-sourced and not having to worry about any of the implications of other people being able to try their code in ways that they hadn't considered. A very long-winded answer to your question, I hope.

[SPONSOR MESSAGE]

[0:16:04.6] Q: For more than 30 years, DNS has been one of the fundamental protocols of the internet. Yet, despite its accepted importance, it has never quite gotten the due that it deserves. Today's dynamic applications, hybrid clouds and volatile internet, demand that you rethink the strategic value and importance of your DNS choices.

Oracle Dyn provides DNS that is as dynamic and intelligent as your applications. Dyn DNS gets your users to the right cloud service, the right CDN, or the right datacenter using intelligent response to steer traffic based on business policies as well as real time internet conditions, like the security and the performance of the network path.

Dyn maps all internet pathways every 24 seconds via more than 500 million traceroutes. This is the equivalent of seven light years of distance, or 1.7 billion times around the circumference of the earth. With over 10 years of experience supporting the likes of Netflix, Twitter, Zappos, Etsy, and Salesforce, Dyn can scale to meet the demand of the largest web applications.

Get started with a free 30-day trial for your application by going to dyn.com/sedaily. After the free trial, Dyn's developer plans start at just $7 a month for world-class DNS. Rethink DNS, go to dyn.com/sedaily to learn more and get your free trial of Dyn DNS.

[INTERVIEW CONTINUED]

[0:18:05.2] Q: That's really helpful. Taking a step back to the question that we started at, which is, how do you find the right people, I think we act as a good example of one engineer creating — Or inventing it, sort to say, and then a whole other team of engineers actually bringing that to the real world and to the outside world. Do you want to talk about why React was so successful in that process even though the original engineer necessarily pushed for it? It still became what it is today.

[0:18:35.4] TO: Yeah. I think the process for building up the team and finding the right people to contribute, the number one thing that you need is optimism, and interest, and excitement. Jordan showed React — Before it even had a name, he showed it to a dozen people, myself included, and everybody had the same reaction. Even me, I was very skeptical. I was like, "Look. I can see the benefits on the developer experience. I don't know how this is going to be

performant enough. I'm really not sure how we're going to apply it to our problems that we're solving here, but yeah. Let's try it out."

There were a couple of other people like Christopher Chedeau and Pete Hunt who provide even more optimism, way more optimism than even I did. When you become surrounded by people who figure out what you're trying to accomplish — And we had a couple of people that just were exposed to this through social connections or connections at work or because of projects they worked on. I think Pete Hunt and Christopher Chedeau both worked on Photos at one point, so they found out about it from a tie that we had there, and they were excited about it and they were interested, and they started playing with it and then they said, "You know what? I need to help. I need to make this happen."

I think that the thing that you look for is you want to bring people onto team that not only have the right skillset, but believe in the vision and believe in what you're trying to accomplish and agree with it. It's one thing to try and manufacture a team out of like, "Okay, we need two iOS engineers. We need one C++ programmer and one designer in this and that." It's another thing to let the team form organically based on the people that are most excited and most passionate. That was part of it.

I think the next wave of team members that came on board were the folks that were excited to be early adapters. Especially if you've been in the industry a while and you've used software that is brand new, not everyone's excited to be an early adapter. The great thing about early adapters is they have an exceptionally high tolerance for instability and bugs and features that are missing and things like this.

If you can find a team to work with, this really easy-going and willing to work with you and willing to not only deliver their product, but also work to deliver the framework itself, they're going to be your biggest advocates. They're going to be the next wave of core team members.

I think, consistently, one thing we've done with the product infrastructure at Facebook is we work with a single team first that is going to use the thing that we're building. We never build software in a vacuum, we never build a thing, and then once it's shrink-wrapped and finished, give it to anyone.

We always build infrastructure, or abstractions, or frameworks, or libraries. We always build them in conjunction with a real user-facing product and we improve both simultaneous. We make the product better through improvements of the framework. We make the framework better though feedback from the product and the teams building the products.

The next wave of people that work on React were people that used React early on. If anyone remembers the original announcement at JSConf in 2013, it didn't go well at all. We didn't communicate well what we were doing, but the people that did hear through that first talk became the next wave of contributors to the project. One of those people's Ben Alpert from Khan Academy who is now — He's been involved in React for a long time and he is effectively tech-leading a lot of the efforts from Facebook's side.

The next wave of people came from, "Okay, if you present about this and you get people that are interested enough to get into the codebase and hack on it and try and figure things out and understand what you're trying to accomplish, that's where the next wave of contributors came from as well."

I think it's definitely a process, and it varies project to project. For react specifically, I think it was this idea of this slow adoption cycle a little bit at a time, get people to buy in and see it and then commit to it. The other thing that really helped with this is was the fact that at Facebook, we really value engineering mobility. We don't want engineers to stay on the same team for forever. We want them to share ideas and cross-pollinate and move across teams and explore different opportunities across the company or really across the companies at this point.

It really was easy for a user of React to become a core contributor to React at Facebook. It was perhaps easier than it is at a lot of other companies. When Christopher Chedeau wanted to switch from the Photos team to the React team, it was like, "I had a meeting with this manager. We set up a timeline, and then it was done." That has really, really helped with a lot of team building around some of these open-source projects.

[0:23:42.6]: Q: That's very interesting. I know that you're getting at that fact that organically building up a team has worked out really well, but if you had to talk about at least a few key roles that must exist for an open-source project to be successful, what would you say they are?

[0:23:59.7] TO: Yeah, that's a great question. I think one role that people might not think about early on, especially in the development of a project, is who is going to communicate about it? Who's going to talk about what it is and what you're trying to do and why?

I worked with another engineer, a very good friend of mine named Marshall Roach. Early on, our relationship, the environment that we worked in was such that he was significantly better at solving a lot of the hard technical problems. He wasn't really as interested in communicating about them to all of engineering, and he recognized that I was more comfortable with that with public posts and with speaking at engineering all hands and things like that.

The partnership that we had was; we would come up with ideas together. I would vet his implementations and review the code and things like that, but then I would communicate about the stuff that he was enabling. That is the only reason that a lot of the work that he and I were able to do together was able to be accomplished. If it was just me, it wouldn't have been able to happen. It wouldn't have been technically the right solution. If it was just him, it might not have happened because it wouldn't have been communicated about actively and proactively and efficiently enough.

One role that you absolutely need is somebody who's going to communicate what you're trying to do, and this can be the same person who's writing code. It can be the same people that are writing code, but you have to have that role. I think that is often overlooked.

The reality of the situation is that as much as the technology that we're building, these are problems of technology, they're also problems of psychology. There's a tremendous amount of, basically, communication that has to take place to enable people to understand and give the project to chance and enable them to get on board with the vision.

The next set of people I think you need are the folks that feel so committed to the project and so committed to the vision. They've really bought into whatever the messaging was that they're

committed to doing all of the things that might not be super glamorous. It's always fun to release a thing but who's going to respond to issues and Stack Overflow questions? Who's going to email people back and answer questions and all these other stuff?

You need a set of people who are so heavily invested in the success of the project that we're willing to do both the actual development work and the testing and fixing and cleaning up and things like that. Also, the software stuff, the maybe sometimes less glamorous stuff of support and repping people up and writing documentation and stuff like that. Other than that, I really do think every team is very different based on the technology itself and who their target audience is and stuff like that. Those are two that I think are really necessary.

[0:26:40.5] Q: Interesting. Particularly on the first role with the messaging, do you have an example of maybe internally an open-source project that didn't do so well in this one, and you think that was a good lesson learned?

[0:27:07.5] TO: Interesting. A project that we perhaps didn't message well enough. I think it's harder once we get to the point of open-sourcing it, but I can certainly think of a several examples of internal projects that weren't broadly adopted internally, because they didn't have a person who was doing that communication.

I can think of also — One example might be that we haven't really communicated with the community, I guess, in open-source as much as Flow, especially when we first started out with Flow. We were building a thing that was going to work in our environment but we didn't really have as much of a spokesperson that was very proactive with messaging about what we're trying to do and almost persistent about it.

It was like, "Hey, here is a thing that we're building. It's working well for us. We're going to continue investing in it." It was almost more of an internal project that happen to be open-sourced and we would work in the community and things like that.

On that project, we might also have not invested enough in fostering a community and answering questions proactively and solving the issues of the community, because there are still so many problems just to solve for Facebook alone. Projects like that, which are open areas of

research and exploration and just truly state of the art in academics, it's really hard to balance trying to deliver value for both us and the community.
Maybe that's one example where a single person didn't really emerge to become what like Tom Dale and Yehuda are for Ember or what Pete Hunt was early on for React. I guess that's one example I can think of.

[0:28:56.6]: Q: Interesting. How has that turned around today? I know Flow is increasingly being adopted nowadays. Just for listeners who don't know, Flow is a static type checker for JavaScript, right?

[0:29:08.6] TO: Yeah. It's being adopted a lot more recently, especially in the React community. I think one of the reasons it started to take off is just because it's gotten a lot better. Eventually, the technology speaks for itself, but the problem is that maybe that process took longer than it needed to, but it's been working exceptionally well for us internally.

I'm excited for — I managed the Flow team for a little while. Even though I don't have programming language background or anything like that, I managed the team for a while. They're just absolutely awesome, super, super smart. They've been able to extract a tremendous amount of value out of our Facebook JavaScript code base. I think that has just translated well to other people as well.

Uber just announced their UberEATS app, which is a React Native app. They also use Flow and some other things. It's good to see other companies adopting Flow and it working well for them, because the reason we originally built it was just to give ourselves more information about our JavaScript codebase and if it ends up being generally applicable in the long-term, that's fantastic.

As for what else has changed, I think, just organically, as more people have tried it and gotten value out of it. The hard thing, specifically, about that project is the JavaScript community is not only bifurcated amongst the different type solutions, like TypeScript and Flow, there's the majority of people in the JavaScript community don't even yet know why types are valuable.

Unless you come from a strongly typed background, strongly typed programming language background, or have some experience, or you might not, at first glance, see why, "It seems like this is more work. Should it be —" It ends up really benefitting you in the long run when you have increasing size of your codebase multiplied by the increasing size of the team working on that codebase. That's when systems like Flow and Hack have really, really shown their stars.

[0:31:03.5] Q: Yeah, I agree. I totally agree. While we're in the process of talking about success of open-source project, I'm curious, do you, internally, have any metrics that you use to track the success of an open-source project?

[0:31:18.0] TO: It's a good question. I think we do have dashboards and things to track the health of open-source projects. If there's a lot of really old, really stale issues, or people getting frustrated, or open PRs, or something like that, we track that kind of stuff. Superficially, people track stars in an informal way. They're, like, "Oh, yeah. We just crossed this milestone." There was a post that we did for React when it crossed, I think it was 50,000 GitHub stars or something like that.

I think that the metrics that we track that we care about are meaningful contributions from the community and maybe —

[0:31:58.3] Q: How would you define meaningful contribution?

[0:32:00.6] TO: Yeah. There's a couple of little different levels of contribution, and I think they're all amazing. Anybody who comes in and just fixes the website, or fixes documentation, is great. Especially if they find things that we overlooked or didn't find, or if they complete some stuff that isn't there. There have actually been contributions from open-source that have — I don't know. Pretty meaningfully affected performance of certain things. Somebody submits a poll request and then all of a sudden, the main Facebook app is faster, or something like that.

We track those types of contributions. I think we just kind of — That's one of the ways that we sell the winds of open-source. That's one of the reasons that the effort, and time, and energy that goes into open-source is worth it. We track some of that stuff. I can't think of any off the of my head on this part here, but I don't think we track too many metrics. I know we have internal

dashboards and things like this, but the metrics that we care about are, “Is the project healthy? Is it adding value to people’s lives?” We track the sentiment and stuff around projects. What do you think would be a good metric for us to track?

[0:33:14.4] Q: Yeah, it’s a tough one, because I think within the GitHub project itself, there’s hard metrics you can track like issues and making sure that there’s a certain time limit for how quickly you response or how quickly they’re closed or how many stale issues there are. I think those are kind of some of the more concrete ones. I think, in general, the sentiment is the one that’s hard to measure.

[0:33:38.2] TO: Yeah, I agree.

[0:33:39.4] Q: You’ve had very successful React Conferences, and each year they continue to get better, but how do you measure that. I really don’t know.

[0:33:47.1] TO: What’s our net promoter score I guess is — There are some things that we get excited about when people do analysis of how likely are you to use — If you’ve used React, how likely are you to use it again? That has a really high percentage, something like over 90% of the people say they’d use it again. We don’t track those things, but we certainly — When they come up, we use it as one more input that we’re either doing something correct, or if there are negative that we’re doing something wrong.

I guess adoption is good. Our intention when we open-source software is never to make it become just, “Oh! We’ve solved the internet.” Yeah, everybody in the world uses this. We’re not even — We don’t even have a goal around, for example, React Native adoption to make it be, “This is the only way that you build apps.” What we do with all of our solutions and all of our open-source projects is this is a thing that solved this set of problems for us, and if it solves those problems for you, we’d love to work together on it.

We’ve had pretty amazing contributions from Airbnb and WIX.com, for example, on React Native and doing different things with React Native. That collaboration couldn’t have happened if those projects were closed source.

To me, the best metric is the meaningful collaboration and adoption. Adoption for adoption's sake I don't think is interesting. I'm really not worried about or interested in every company on the planet using React. There's actually cost associated with that. There's more support cost and more maintenance burden and all these other things and trying to make it do things that it wasn't designed to do.

Tracking adoption, meaningful adoption, where you're getting input and you're getting new use cases that help you test your hypothesis and you're getting meaningful contributions back. That type of adoption, I think, we do track and we do care about.

[SPONSOR MESSAGE]

[0:35:45.6] JM: Dice.com will help you accelerate your tech career. Whether you're actively looking for a job or need insights to grow in your current role, Dice has the resources that you need. Dice's mobile app is the easiest and fastest way to get ahead. Search thousands of jobs from top companies. Discover your market value based on your unique skill set.

Uncover new opportunities with Dice's new career-pathing tool, which can give you insights about the best types of roles to transition to. Dice will even suggest the new skills that you'll need to make the move. Manage your tech career and download the Dice Careers App on Android or iOS today. To check out the Dice website and support Software Engineering Daily, go to dice.com/sedaily.

Thanks to Dice for being a loyal sponsor of Software Engineering Daily. If you want to support the show and check out some new career opportunities, go to do dice.com/sedaily.

[INTERVIEW CONTINUED]

[0:37:06.6] Q: That's interesting. I never thought about meaningful versus not meaningful adoption. I guess you're talking about it in terms of — Why wouldn't you just want everyone in the world to adopt React? Is it just too much pressure on the organization?

[0:37:21.0] TO: No. There are other solutions to problems out there that are amazing, and I don't think that React solves everyone's problem everywhere, nor do I think that it would be as good of a tool as it is if it attempted to. I've played with lots and lots of other frameworks that are unrelated to React, but that I think could be influenced by React. I don't know. I don't think that — I guess there's this tendency to always want kind of more, more, more, more influence, more adoption, more subscribers, more stars. Everybody kind of even tweets about this stuff when they have a new — It just feels good that you've been validated once you have a kind of high profile client or adapter, but I don't think the world would be very interesting if everyone in the world use React.

[0:38:15.4] Q: Yeah, that makes sense. I guess what's interesting is, then, there's the other side which is the ecosystem is completely fragmented and you have a fragmentation problem. Particularly in JavaScript, there's seems to be an issue where it's either — It's just like so many different solutions for the same thing, or do you want one really good solution? That's an interesting balance to have.

[0:38:43.8] TO: When I think about that question about the fragmentation and maybe even bifurcation, the concrete downsides of that in my opinion are a less of a willingness to share ideas and what I would call siloing and closeness.

If we can solve closeness and siloing and enable more engineers to collaborate, then I think it is a worthy goal to say, "Yeah, we want more people using React or using something, anything." If we can find some webs that gets more engineers that wouldn't traditionally be talking to talk, I think that is absolutely valuable. I also think that you do want some amount of fragmentation, because that's how new ideas form. If we had just used what everyone else was using when we created React, we literally wouldn't have been able to create React.

You want experimentation. It's not so much like, "Let a thousand flowers bloom," but it's kind of similar, it's, "Let's innovate independently and then come back together and share ideas and share code wherever we can and hack. Let's share languages. Let's share programming languages. Let's share tool chains and tooling and things like this."

[0:39:56.3] Q: It's a very interesting topic, because I think as you open-source technologies, one thing that naturally happens is people create their own solutions off of what you open-source, because they need some part to be different. Even within organization, they'll have their own alternatives of React, or whatever it is.

Have you thought about how to not only encourage that, but also encourage them to be open about what their ideas are so we can all come back together and create an even better solution together instead of the silos?

[0:40:29.7] TO: Yeah, this is a really hard problem. How do you get folks whose livelihood is dependent on them having a better solution? How do you get them to share their solution back? I don't know the answer to this, but I think that the answer involves more communication and collaboration and us fostering an environment where we have as much inclusion as possible.

One thing you'll notice about the React community is no one will ever bash or complain about any other frameworks or libraries or anything like that. If they do, they'll be basically, not scolded, but they'll be reached out to by somebody from the React community that says, "Hey, that's not cool." We celebrate wins. We celebrate new things that happen, new developments.

Yesterday at Ember Conf, the Ember team released Glimmer. I can't wait to play with it. We should celebrate each other's accomplishments and create an environment where people are more likely to want to share their solutions, because they're novel and new, and not because they're a strategic advantage for them or a strategic sort of source of — I don't know — Proprietary worth. If that makes any sense.

I think it involves all of us creating an environment where we celebrate the developments of others rather than — I don't know — For lack of a better way of saying it, crapping all over their ideas. I basically don't read Hacker News or Reddit anymore, because I can't stand to see people hating on everybody else's ideas.

[0:42:01.4] Q: Yeah, I totally agree. That's interesting. I kind of want to transition off and move into talking about contributor policies and rules. This is a very interesting thing for me, because I think React follows a more close core team approach, whereas Ember, which is another

frontend framework, follows a more open RFC type approach. I was wondering if you can quickly, in a couple of sentences, just explain to listeners what the closed approach versus open RFC approach is and why React has chosen to go with the former and not the later.

[0:42:36.4] TO: Yeah. Real quick, just to compare and contrast a little bit. An RFC approach, I think you sort of accept any type of proposal from anyone in the community. If it garners enough support, it will be considered for inclusion by the core team, and maybe some RFC say, "If it garners enough support, it will be included. There's no discretion."

I think the model that React has, which I think you referred to as closed team, or what did you call it?

[0:43:05.7] Q: Closed core team.

[0:43:07.1] TO: Closed core team. This is a purely an implementation detail and a side effect of sort of the way that the projects kind of grew organically. I think what you mean there is the development of the framework is sort of at the discretion of the core team, the people that are stewards of the project and are maintaining it and working on it. Most of them, I think at this point, probably work at Facebook, although there's a handful of non-Facebookers.

I don't necessarily think one approach is fundamentally better than the other. I think, at the end of the day, someone needs to have a long-term vision for where the project wants to go in their head, and whether that is a group of contributors, a group of a closed core team, or something like that, or it's a single person. A lot of projects just have a single visionary. Somebody has to be kind of responsible for deciding what this should be and what it should not be.

It's really hard for people to list out non-goals when they're thinking about planning and things like this. If everything was allowed to be everything, then our software wouldn't be valuable. Everything would just be huge frameworks and they would do everything. You have to have somebody who's sort of in charge of the vision and where we want to go.

After that, I think the process for getting changes into the framework, or getting changes into the project, to me, it's kind of do whatever works. I think the reason that React has the approach

that it has is probably because the folks that are investing directly in React, a lot of them work here at Facebook and they're trying to solve real problems for real engineers that happen to also be at Facebook while simultaneously meeting the needs of the community.

We tend to — While we would happily accept any sort of RFC process and any proposal and things like that and we will evaluate it, we tend to focus on the roadmap that we've planned out for the past six months, to a year, first and foremost. Then, the other things, like the other parts of the — Anything that Elsa had suggested is, "Oh, yeah, Definitely. We want to get to this." We'll add it to our list of things to focus on in the future when we do our next round of planning.

Right now, we're driven by — We have a very specific goal. If you contribute to that goal, it is very likely that your contribution will be not only accepted, but appreciated very much. If you're working on something else, it might take a little bit longer to get it than — I don't know if that answers your question about the RFC process versus the closed core team. I guess it catches me a little bit by surprise, because I don't necessarily — I don't think we're actively trying to be a close core team. I think it is just a sort of implementation detail of how the team is currently structured, but we would be happy to revisit it for sure.

[0:46:03.5] Q: Yeah, interesting. I think the RFC process if interesting, especially Ember's, and I'm curious to see, I don't know what other projects have that similar process. Even the communication of that makes it seem a little bit more opening, if that makes sense.

[0:46:22.5] TO: Yeah, that makes sense. I think if folks feel, right now, they aren't allowed to or aren't able to contribute to React because of its sort of model for development, then this is certainly something that we should definitely consider changing it. I don't think anyone on the core team is actually thinking about this, so I will bring this up with them.

I do love the RFC process, at least the couple of RFCs I've read. I haven't read an Ember RFC in a while, but there's also — I think the Graphic UL team tried to adopt a Graphic UL RFC process, and seeing some of that stuff is great. We had a proposal for JSX 2.0 that I think is still under consideration, and we'd love to do when we can find the time here. I don't think it's actively trying to be close-source, but maybe we can make it even more open.

[0:47:10.9] Q: Yeah, interesting. Do you think it's a requirement for companies to dog food their technologies before open-sourcing it? Because, for example, there's — I'm not sure if this is true, but I've heard React Native is not used in large parts of the Facebook app just yet, but it is open-sourced. Do you think it's a requirement for companies to dog food it first before they open-source it?

[0:47:37.4] TO: Yeah, it's hard to judge the scale, because we have sort of — I don't know — Many dozen engineers writing React Native code every day, and we have something like 400 screens across a couple of dozen apps that are using React Native. Because people don't see many apps that are entirely written with React Native and they don't see, "Oh! The Facebook main app doesn't use React native in newsfeed, so React Native is not used by them."

It's kind of a little bit — Or maybe our messaging is just wrong on this, but we are using it very heavily internally, but the scale at which we operate, when you have so many engineers, it's like if all of those engineers aren't using React Native, is it being used internally?

We have literally hundreds of engineers that write React in React Native every single day. I think this is pretty critical. I actually think we won't open-source something if we're not already using it, or if there's a very special extenuating circumstance. The reason for this is because we've done it in the past. We open-sourced a couple of things that we weren't fully rolled out on in production and we're like, "Oh! This doesn't work." Meanwhile, a bunch of other companies adopted that technology, and then we left them in the dust when we went and pursued something else.

For me, as a manager, whenever somebody comes to me and says, "Hey, can I open-source this thing?" My first thing that I'm looking at is, "Okay. Let me talk to the people that are already using it and make sure they're happy with it," because if they're not happy with it or they don't exist, the project that you're releasing isn't guaranteed to be maintained.

A lot of companies will sort of build something first, open-source it, and then try and get internal adoption through open-source, and I really don't agree with that approach. I actually think this is one of the reasons that our open-source projects have been at all successful is because they are relied upon heavily by Facebook. We have 1.8, or something, billion users hitting React

every single day, and we use a nightly of it. You could rest assure that we're not going to easily break your app because we always have to find an incremental path from where we are to where we want to be, because it is in production.

Yeah, sort of feel pretty strongly about this, that I recognize that it should be okay for companies to open-source things before they're using them, but I personally won't really kind of allow that at Facebook.

[0:50:07.2] Q: Yeah, that makes sense. I think the — Particularly talking about the whole release upgrade path, I think that's super important. If you're not testing it internally, it's hard to know once it goes out in the wild.

I guess I kind of want to move on to some of the lessons learned along the way as you've done open-source. Maybe because you're more focused on the React team, what are maybe some mistakes that you made along the way or things you think you could have done better?

[0:50:37.4] TO: Yeah, one thing we could have done better was skip to the first talk where we presented what React was and then just have Pete Hunt give his second talk from React Europe.

[0:50:48.6] Q: We'll put that talk in the show notes.

[0:50:51.1] TO: Yeah, that's perfect. Yeah. Pete Hunt gave a talk entitled *Rethinking Best Practices,* and the title was actually adopted from a Tweet that was sort of a jab at Facebook after the initial presentation of React. The reason was like we just really didn't communicate about the problems that we were trying to solve very well, and people got sort of superficially hang up on either Syntax, or the idea that we were going to combine HTML, and JSS, and JavaScript, "Oh my gosh! This is the worst thing ever. I can't believe Facebook. They've been under a rock for so long."

One thing I think that I would do different if I could go back and do it again, and maybe this actually — Maybe I shouldn't say this, because I think as a result of it not being super — Not having a lot of people super excited about it. We have this slow burn where we were able to get

a couple of key people onboard and make it a lot better before we have to start facing the influx of adoption about a year later. That's kind of one thing.

I think another thing is React was kind of the first project, or one of the very first projects from our new open-source regime, if you will. Basically, before that, it was a lot of stuff that wasn't maintained and we didn't really — It was okay, and it sat there, and it wasn't super active. With React, it was something that we were actively developing and making better.

Maybe I wouldn't change that messaging as much. Maybe it was kind of, "Yeah. Look, we announced this thing. Nothing to see here. We'll call you in a year." I would like to have been more open about what went into it. I think we were still learning as part of this new open-source initiative, and I would have like to be more open and communicate about what problems we were solving.

I really kind of like — I hate these very splashy launches that claim that they've solved everything. I always wanted our messaging for our open-source program to be pretty consistently, "Hey, here is a thing we built that solves a problem for us. Try it out. Let us know if it works for you. We'd love to collaborate if so."

Often times, our messaging was probably either not that or maybe too grandiose or something like that. One thing is just making sure our messaging is always consistent and our motivations are always consistent. Yeah, communication and sort of being really open about what we're trying to do and why and fostering a community of even more contributors early on I think would be good.

Another thing I think I'd do differently is just like — No one really knew for a really long time how to contribute to React. It's super intimidating. The codebase is kind of hard to make your way through, and we're really improving that with Fiber. The new codebase is a lot more approachable. Even though these underlying sort of architecture is a lot more sophisticated and complicated.

[0:53:42.9] Q: Maybe for listeners who don't know what Fiber is, do you want to give them a quick —

[0:53:46.4] TO: Absolutely. The short version is we rewrote React and we did so in place. The idea is that we're going to enable more advanced scheduling over the next couple of years of work that needs to be done rather than the traditional model of just, "Oh! Just render everything."

Yeah, we're really excited about Fiber, but along with it comes a new set of contributors, and a new codebase, and it's fully Flow typed, and it's a lot more approachable. One thing I might do differently is, early on, document how to actually contribute to React. How did you think in React? Especially since it was such a novel and new concept at the time. There's so many ideas that we now just take for granted. We're sick of hearing about them all day, like Virtual Dom. It's like, "Please. Don't talk about the Virtual Dom anymore."

At the time, we didn't really document what that meant. We didn't really document what we were going for with that. I would sort of, if I could change something, I would say, "Let's go back and create an environment where it's easier to get new contributors on boarded, especially from outside of Facebook." I would really love to have an even stronger and larger community of actual core contributors, not just around the ecosystem. The React ecosystem is unmatched. It's amazing. We have so many people building so many awesome things that either work with React, or for React, or on top of it.

I would like to have even more core contributors that are comfortable making decisions on behalf of the framework. Right now, I think, because it's just been so intimidating to contribute to in the past, we don't have a ton of core contributors that feel empowered to make decisions on behalf of the framework.

[0:55:21.0] Q: Interesting. That's a very good answer to that. I guess sort of in similar around of things, kind of going off of some of the lessons you learned, do you have recommendations or tips for people, or companies that are newer to open-source, best practices, or things to avoid, or things to make sure you do.

[0:55:43.5] TO: Yeah, we touched on a couple of them. One of them is at no point should you claim to have solved everything. If you solved a problem and you solved it well and you're

happy with it, ask some others to verify it. Work with one single client. Find one person whose life you're going to make better. Find one team whose product you're going to make better or find that first client that's going to help you vet your hypothesis a little bit.

Then once you have that, then you're messaging — I think all messaging for all open-source projects should be relatively consistent. It's like, "Hey, we faced these problems; X, Y, and Z, and this thing helped us solve those. Here is the tradeoff, because everything has a cost, here is what it cost us and we believe that this is the right tradeoff. Let us know what you think if it works for you or if you have any other ideas."

That would be my first thing, is like just stop coming out with the better React, or the better angular, or the better this, or the better that. Say explicitly what the problems are that you set out to solve. Say how you solve them, and then open-source your code and offer it up to people that can vet and help prove your hypothesis a little bit. I feel like there's a culture of everybody wants a few minutes of fame. They want something to go viral. They want some top post on Hacker News, or whatever it is. If our messaging was a little bit more humble and a little bit more honest about the problems that we're solving and the tradeoffs associated with that solution, I think that would be good for everyone.

Another thing is just trying to bring collaborators in early no in an open way and trying to create an environment of inclusive contributors. I think most frameworks, and libraries, and products, and things I've seen that are open-source are pretty good about this these days.

[0:57:36.3] Q: Yeah, interesting. That's really cool. One thing that is less applicable to, particularly, open-source a Facebook — Open-source in general, burnout and abandonment is such a common theme amongst open-source projects. When it's a small one that one developer put out and it's not used by that many people, maybe it's not a huge issue, but once it kind of goes in popularity, that becomes a huge issue.

I guess the problem though is that we're human and we — A lot of engineers, particularly, they like to build stuff. They like to solve problems, but they're not necessarily into the whole maintenance part of it, and the support part of it. It's completely natural for someone to have found excitement in solving the problem, solving the problem through the open-source

technology, but then losing interest later on. How do you kind of manage this within Facebook, as well as how would you — What are some recommendations you might have for just the general open-source community around this?

[0:58:38.5] TO: This is very real. It happens absolutely all the time, especially as a consequence of popularity. If you have a project that only a couple of people use and you build a very tight net, very small-ish community, and they have a problem, you're probably going to be — Once in a while, you're probably going to be motivated to say, "Yeah, let me jump into that," and we see that often.

Once something gets to the scale of — Here's an example, like Babel. Sebastian McKenzie, he ended up joining Facebook and he's literally the only thing that he was responsible for doing was working on Babel. Not even just for Facebook. Facebook was using it, but just for the community, "Well pay you full-time. Just continue working on Babel. Keep carrying your vision out. Here're a couple of things that are really important to the community. Here're a couple of things that are important to Facebook."

With the release of Babel 6, I felt like he had a very strong vision for it. We kind of all agreed. A lot of the people in the Babel community agreed with the vision. Then, people who started using it, and it required that when they wanted to adopt Babel 6, they make changes to their code, just started being exceptionally unreasonable to him and super mean. As a result, he just really didn't want to deal with that community as much anymore, and so he wasn't as responsive on issues as he'd been. He wasn't as responsive on PRs and stuff like that. He just basically burned out from the sort of treatment. We very quickly found another awesome set of projects for him to work on.

The reason that Babel continues to thrive and continues to be so valuable today is because it wasn't relying on a single person to maintain it permanently going forward; the current Babel core team continuing to iterate and improve and innovate. The thing that you have to do is long before you even are at risk of burning out, you have to build up a set of people that also believe in the project and are going to help you steward it. They're going to help you maintain it and find a good home if something happens to you.

I don't think it's just about burn out, that's not the only risk. We also — As a company that values engineering mobility and engineers moving from team to team, we have to be prepared for — And engineers just moving off the team that maintains Presto, or an engineer stops working on Hack.

We're constantly bringing new members into the team and making sure they're kind of motivated and helping and things like that, and it's not just about employees either. It has to be a community. One pretty easy thing you can do in order to get people motivated to help out with closing issue and merging port requests and answering questions, is just recognize them for it. Just appreciate them for it.

The first project that I ever became a core contributor for was called MooTools. We've talked about it in the past. The first thing that happened was the creator of the project thanked me for answering a bunch of questions on the, then, like PHP BB Forums, like some bullet and board forums, where I was just answering questions for people, because I've been through this stuff and I knew the answers. He made me a moderator of the forums. He said, "Just wanted to call this out. Thank you so much for your help. Now, you're a moderator, and I trust you to kind of help scale this." I took that responsibility so seriously.

I was young and it was my first project and I didn't really know much about open-source at that time, but if you find somebody who's adding value, recognize them for it, appreciate them for it. Let them make that part of their identity. Give them commit access. Give them admin access, or whatever, and enable them to utilize the trust that you've sort of emplaced in them. Does that make sense?

[1:02:24.2] Q: Yeah. That's really, really good advice. I've heard this from a lot of authors, and they kind of just get sick of doing a lot of work and not getting appreciation in return. I can see immediately how that really affects you emotionally. Open-source is 50 — Or maybe even 100% emotion. There's a huge motional component to it, because a lot of the times you're doing this in your spare time when you can be doing other things.

[1:02:54.1] TO: You said something about appreciation as well, and I just want to hit that point for one second. The best way to be appreciated for your work is to appreciate other people for

their work. The best way to get recognition for your work is to recognize other people in their work. All too often, our default mode is, I'm using a thing, it breaks, you owe me. It's like, "Look. It was free.

The person put their time into it. How about let's reframe this? How about instead of fix my bug, you say, "Hey, I just noticed that thing broke. Is there any way that I can help?" How about offer up help and offer appreciation and you will get appreciation in return and you will get recognition in return? These things are side effects. They're not the goal.

[1:03:37.7] Q: Yeah. I think there has been some communication recently about — People are being a little bit more open about their frustrations in this area. I think it's becoming more aware. Are people always aware of this problem? I think for a long time, people weren't even aware of the treatment that some of the open-source authors were facing.

[1:03:56.5] TO: Absolutely. If your one of those authors and you're getting this nasty comments and posts and email, you don't want to highlight that. You want to just push it down by default. You're just like, "Oh, I just want to forget about this." Yeah, it's good that it's being brought to the forefront a little bit more.

[1:04:12.3] Q: Yeah. I think we've gone through a lot of the questions I wanted to go through, but I kind of want to end on a future-looking note. Are there any, either, internal projects that you're really excited about that are soon to be open-sourced, or just areas of open-source technology that you're super stoked about?

[1:04:31.1] TO: The answer is yes. As to how much of it, I'll talk about it. I think I won't get into specifics, but I will say, sort of every year, we have our annual developer conference; F8. At that developer conference, we have a lot of engineers come and talk about the things that they've built, and inevitably. Some of those things will be public. I'm looking forward to F8, which is April 18th and 19th this year.

Yeah, there's a bunch of other things that we've been working on sort of in the open. There're developments for various projects. There are new things coming to React, React Native, some exciting stuff coming to GraphQL, exciting stuff coming to Flow. There's a lot I could tease

without going into specifics. We're not slowing down in anyway. If anything, we're probably even maybe speeding up a little bit. We have more engineers building more open-source stuff, and lots to be excited about.

[1:05:23.5] Q: Yeah. Are there new areas you're focusing on? For example, machine learning, or —

[1:05:28.9] TO: Yeah. As a company, Facebook, especially in VR, machine learning, AI, all these things. There's a ton of research and development happening there. I'm a little bit removed from them. I'm not working on any of that stuff day-to-day, but yeah, keep an eye on the space, because there are some really, really interesting and exciting stuff happening — ML, AI, VR, and all the other acronyms as well.

[1:05:51.8] Q: Yeah, awesome. Thank you so much, it's been a really, really fun interview, and I'm sure that listeners will have a lot to learn from. If they want to contact you, where should they reach you?

[1:06:02.5] TO: Yeah, Twitter is probably fine. Ping me on Twitter or something like that.

[1:06:05.4] Q: What's your handle? We'll include it in the show notes.

[1:06:08.3] Q: Sure. Tomocchino, my last name is pronounced Occhino, by the way, for anybody who cares. Yeah, ping me on Twitter. You can also shoot me a message on Facebook, on facebook.com/tomo. Yeah, this has been fun. It's awesome. I always love kind of talking about this stuff, and I really appreciate you having me on.

[1:06:26.1] Q: Awesome.

[END OF INTERVIEW]

[1:06:31.2] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning

from each other. Check it out at symphono.com/sedaily. That's symphono.com/sedaily. Thanks again Symphono.

[END]