EPISODE 01

[INTRODUCTION]

[0:00:00.3] JM: Dan Abramov, you are the co-author of *Redux*, you're a member of the Facebook React Team. Welcome back to Software Engineering Daily.

[0:00:07.2] DA: Hi.

[0:00:08.2] JM: Your path to working at Facebook full-time was through the React community. Why did you start working with React?

[0:00:15.3] DA: I needed to write an app in JavaScript. Before that, I was working a small startup at the time and we were working on an iPad app, but then right in the middle we decided to do a web version instead. We needed to make this pretty complex, single-page app. I don't think either of us even – there were three or four people on the team at the time and I don't think either of us knew how to do it.

I think we started with backbone, but we bumped into its limitations. It was pretty difficult to create dynamic, highly dynamic user interface. One colleague showed React to me. At first, I dismissed it, but then he was like, "No, you should really try. It is good." I implemented a – it so happened at the time, I was implementing a like button. It's literally the button that you can click, it says you and other friends liked this post.

I tried React on this small like button. I was able to express all the UI states that I wanted in a single evening. I just checked in React into our app and then gradually, we ended up gradually rewriting the whole app in React alongside as we were working on new features, we managed to ship new features and rewrite. React gave us so much productivity that we could both rewrite things in React and also ship new features at the same time.

At the time, it was really a pretty early time for React. I think we adopted it in 2014, so that was a year after its open source release. A lot of the ecosystem was still missing at the time. There

wasn't even React router back. There was another library called React Router Component, I think, which didn't end up being popular.

It was a pretty early time, and so I forward a lot of problems. I just had to figure out my own solutions, like how do you set the document title? How do you sync React with backbone and all this stuff? I ended up creating some libraries just to solve our own problems as they came up. Then that was at the same time when React ecosystem was exploding and I think a lot of people noticed those things. That's how I became somewhat visible in the React ecosystem.

[0:02:58.6] JM: You joined Facebook in 2015 to work on React. What was the structure of the React team at the time when you joined?

[0:03:08.1] DA: You mean like, who exactly worked there?

[0:03:10.7] JM: I guess, the management structure. Because this is an open source project and you're joining to work on an open source project full-time. Did you report to people, or what was the structure of the team?

[0:03:23.3] DA: Right. As far as I remember, Tom Occhino was managing the team at the time, so I reported – just like everybody on the React team, I reported to Tom Occhino. Tom Occhino is still at Facebook. He's managing the React org, which includes the React team and a few other teams. Yeah. Now we've went through a few manager changes. For some time, Christopher Chedeau managed the team, then Sophie Alpert managed the team. Now Zhang is managing the team. It's been a few different people over the time, but they reported to Tom.

[0:04:04.2] JM: What have you learned about having a open source product team managed successfully?

[0:04:12.6] DA: That's a pretty broad question.

[0:04:15.4] JM: It is.

[0:04:17.1] DA: I think there is a lot of stuff that I learned. I think Facebook's approach to the open source is different from some other companies and it has both upsides and downsides. I think one of the guiding principles for what we do and what I think why React managed to be successful is that React is not a vanity project, right? Facebook is not a developer platform, at least in the front-end space.
It doesn't actually – Aside from hiring, I guess, and recruiting, we don't actually have strong incentives to encourage other people to use React. React is primarily driven by the needs of the products that we develop. Because we develop such a wide range of products, it's like when you think of Facebook, you probably think of I don't know, the newsfeed, but React actually didn't come out of that. React came out of the ads.

Jordan was walking on I think ads creation. This is one of the most complicated front-end apps that I've ever seen. It's many forms connected to each other with many steps, with validation and all these targeting previews and dropdowns and maps. Any component, you can probably find it there. Jordan was trying to simplify that code and that's how React came to be.

It gradually spread its presence over different parts of the UI and it's been used more and more in the main consumer side apps. I think just being driven by the product needs and having a different set of products helps ensure that by the time that we release an API, or make a change to React, it is actually – it can answer a broad range of use cases. They tend to map to what people externally want as well.

It goes through a lot of dogfooding. Also including, because Facebook itself runs on Facebook, so all the work, communication happens to workplace and workplace messenger, which are pretty much Facebook, but branded for companies. If we introduced a bug in React, like our own chat is probably going to break, so we'll probably notice that. I think that's also why React is relatively stable and I think that a lot of people are using React to appreciate.

[0:07:03.3] JM: How is the React team managed today, if you don't have KPIs and other hard metrics? I mean, surely if somebody pushes a bug and it starts to impact Facebook for work, yeah, that's a problem. Are there any management structures that somebody who is listening to this and trying to build an open-source team might be able to take away?

[0:07:29.1] DA: I think React team is pretty unusual in that sense, even compared to most Facebook teams, because usually when you build a – when I build a product or some piece of infrastructure, that's well understood. Maybe some new cache and layer, or some new network thing, or some database thing. You usually have a well-defined criteria for success, right? You can say, maybe we increase the number of posts by 15%, because the feed flow is faster, or something like this. There is usually some metrics that you can track over time and use that as a success criteria.

With React team, I think we have passed the stage of early – easy improvements, easy wins. A lot of what we're currently working on is our longer-term projects that involve a lot of research. In many cases, the results, they're not necessarily easy to quantify, or even it's not easy to – a lot of React apps you do some data fetching, right? You want to fetch something from the network, you want to store it, maybe in component state, you want to show some loading indicator while I do that. It's I don't know, maybe 50% of the application code and some apps can be just data fetching and management logic.

It's not something that – you can already do that, right? It already works in React. We could say that this is a solved problem. Then if you look at this problem more holistically, you can notice that there's a bunch of boilerplate code. There is suboptimal user experience, where you might see a cascade of spinners flashing everywhere as the data loads. Because you express everything in this component, so you can see this fragmentation in how the changes are presented to the user.

If we could handle this on the library level, we could have a better user experience and we could have a better developer experience. It doesn't mean necessarily that it's impossible to do today. You could hack around those things in product code and people to you. A lot of code is just hacking around things. What we try to do with React is we try to figure out the idiomatic solution, so that the one that doesn't need to change later, the one that has the best trade-offs for this use case.

Then once we solve a certain problem, we want it to remain solved. I think that makes it0 challenging to say, like coming back to your question about the how do you measure success and how do you manage that? I think that makes managing it very challenging, because in

many cases, it's like managing a research project. You don't really know if something is going to work out or not. The best you can do is to create an environment where people can actually do that research, where they don't feel rushed, or they don't feel that this research is useless, but also find a way to not let them get – to not let them go into a completely different direction from what the product needs.

You need to keep them close to the product people and keep the conversation flowing both ways and make sure that they see real scenarios and that they try their solutions in real products and get feedback from that. Of course, you also need expectations. The product side might depend on some API. It's not clear if that API is going to work out and you need to ensure that there is understanding on both sides on what is in each stage. I think it's a mix of many things. It's finding – mostly in finding the right balance between the research, the dogfooding and actually implementing things and shipping them.

[0:11:44.0] JM: You're describing a distinct environment, because to some degree it is this research project, this long-term research project where you need to have some determinism around where the project is going, what you want React to look like in five years or 10 years. To another extent, it's like a startup where you have customers that are already using your product. There are thousands of applications that are using React today and many of the developers who are using React are interfacing with the core team through open source platforms like Github. How do you, you personally, or I guess the team as well, what's the balance between interacting with the open source community and thinking longer-term talking internally with your own team?

[0:12:37.9] DA: I think you have to do both at the same time. It's just work on different threads. I think you need to have some vision. There is a person, Sebastian, Sebastian Markbåge on our team who has all these ideas about longer-term projects that we could do for the next five years. I think we were very lucky to have him. I think a lot of his ideas turn – some of them didn't work out, but a lot of them turned into actual things that we shaped. The react 16 rewrite, which we shipped I think a year ago, maybe two years ago by now, was motivated by some of the research that he did and that Jordan previously did.

I guess, Jordan the creator of React is the first visionary for React. I think it's important to have a person like this who has a long-term vision, but then he also need to talk to the product people and talk to the open source community. I think talking to the open source community is something I try to do. It's not technically my job description. I'm just an engineer working on React, but it also serves as input and it I guess it helps find the problems with the API and the edge cases, sometimes earlier than we find in our own products. It's a really good source of feedback. Can you repeat the question again?

[0:14:09.5] JM: Well, you did basically answer it. I mean, I was hinting at the fact that there's a combination of long-term research that you have to do, but also short term analysis of your current "customer base," or I guess user base, people who are using React components. It just strikes me as there is a balance that you have to hit and is probably not straightforward, but probably also hard to describe, because it is like you said, you're doing these two different things on separate threads.

[0:14:41.3] DA: Yeah. I think, like the threads analogies are really useful one. You can think about it – we don't really talk about React as a single entity as much. We usually talk in terms of projects. We give them funny code names. They all start with the F letter. There was the fiber rewrite, there's a React fire project, there's React fabric. The code names represent the fact that we don't always know what those projects actually mean, because we're still researching them.

Usually, there is some idea that we want to explore, but then eventually we get into a phase where we're confident that the design is solid, that we can try to implement it, dogfood it, try it and then we see okay, it seems this works, it seems this is a good idea, and we can get closer to shipping it. That's when we can drop the code name and it just becomes React again, or it becomes some specific feature, or a specific release like React 16.

As the project gets through – each project gets through those stages where at first, you just have a hunch, like maybe this is a good idea, it's worth trying. It's when you explore the designs. Then when you actually start working on it, what do we try to do is we try to focus on a single client first. This client can be – usually it's somebody internally, because we don't want to release an experimental API and turn the community on something that is not actually good. We try to find some volunteers, some teams that are willing to try any unstable API like at Facebook.

At first, we always pick just one team. Even if 10 teams want to use it, we just pick one. We work with them to ensure that their use case is handled well, that this API actually works, that is fast and that they feel happy with it. After we've worked it out with one team, this is when we pick the next team and we make sure that maybe they need a different option, maybe their use case uncovers a flaw in the API that the whole idea might get scrapped because of that, or maybe we find a way around those and make a better API.

We gradually take on more clients until we say, “Okay, let's just try to use it at Facebook.” Any team at Facebook can start using it and they give us feedback. This is the point where if it goes well for a month or two, then we feel confident. Yeah, this can be a stable API in the open source, and so we release an alpha, or a beta in the open source, start writing documentation and we get another round of feedback, which might also lead to some changes in the API in the idea.

Each project goes through stages and it gets more and more concrete, but it's important that we start by focusing on a small user base and try to make that user base happy and then gradually expand.

[0:18:03.3] JM: You have said that in the early days of joining Facebook, you didn't have a work-life balance. It sounded like that was because you were enjoying your work so much. Can you tell me about finding work-life balance?

[0:18:19.8] DA: I don't actually remember the time I was referring to – I mean, I think I'm still struggling with work-life balance sometimes, which is mostly self-imposed. It's not even the work itself that eats into that as much as just answering people's questions on Twitter and others, or github or social media. It is pretty difficult for me to completely tune out of that. I do try to not look at Twitter in the evenings, although it's again, I'm not doing – you caught me in the moment where I'm not doing that particularly well. Especially I think as I get a bit older, I'm 20 – I think I'm 26 maybe. Yeah.

I used to be able to argue in the evening easily, but now I'm like, if I – I don't know, if somebody is really negative and it's even 8 p.m. or 9 p.m., I try to reply in a calm way. Sometimes it still

gets to you. Then I have a bad night's sleep. It's just not worth it. I'm trying to cut down on my social media usage as much as possible.

In terms of work, I don't really think it's a problem now. I don't take my work laptop at home, so I couldn't fix anything even if I wanted to. I'll admit that I do read the conversations in the workplace chat. It's challenging because my teammates are mostly in America while I'm in the UK. Just as I'm about to start, I don't know, to watch Netflix, or do something in the evening, there is an interesting conversation and sometimes it's hard to resist and not jump into that. Also, I've been trying to do that less in the past three months.

[0:20:28.9] JM: What are the biggest problems with how popular open source repositories get managed today?

[0:20:34.5] DA: You mean in terms of what are our biggest – what do we do badly, or what –

[0:20:41.9] JM: From the point of view of the company, from the point of view of the developers who are consuming that open source repository, from the point of view of the overall moderation and the social interactions that take place on these open source repositories. What are the problems with the ecosystem?

[0:20:58.3] DA: I guess, I'm not really qualified to speak as to what the issues for the company are, or even what the issues for the users are, because I haven't really – I haven't been a consumer of a large open source project for a long time. I don't really remember what it was like. I think that things that – I think that makes it challenging for us – I'll just be selfish. I'll tell you what is hard for me. I think the thing that becomes challenging is we – every single thing that we do is heavily scrutinized, which is fine. It's a popular project. We welcome the feedback and stuff like this, but sometimes it just gets overwhelming.

Imagine that I don't know, you were working on an app and every single commit would be commented on by thousands of people. It's just a bit weird. I want to say that this is exactly what happens. We don't get a thousand the comments on every comment, but sometimes it means that it can be difficult to manage the communication and to own your own narrative. Sometimes we want to make a change, like we don't do big changes in React without a good migration path, right? Because we have by now, I think probably a 100,000 components in Facebook,

React components. Maybe a bit less, maybe I don't know, 80, 90,000, especially if you combine both the web and React Native repositories.

We usually have some migration path, because we are just a team of 10 people basically, against all those thousands of components. We can't leave somebody behind. We don't have multiple repositories, where everybody picks their own React version. No. There is a single React version that is basically React master from github and that we sync every two weeks. Everybody is on the same person of React. If we make a change in React, it's on us to make sure that those thousands of components actually keep working with this new version.
We take the upgrade paths very seriously. Then it can be difficult to communicate that to somebody who just adopted React. They don't know us and they don't really have a good reason to trust us. That's completely understandable. How do they know that we're not going to just move their cheese?

What is difficult is we need to make sure that our messaging is always very clear about what guarantees we provide, or APIs are stable, what APIs are unstable, how we're moving forward without leaving anyone behind and stuff like this. Sometimes if we fail to communicate that clearly, then somebody else will do that for us, because it's a very visible project.

If we release a feature and we may even document it, but if we don't explain it on Twitter, or in a blog post, or probably in both combined, then somebody else will explain it for us. Sometimes they might misunderstand either the feature itself, or the intention, or the upgrade path. Once somebody does it, then the misconception spread. Because React is such a visible project, literally you can push a release and the next day you will see 20 medium articles explaining what it is and why it matters. Keeping track of everything that people talk about and helping manage – helping explain our story and why we're working on these things and what those things really are is pretty difficult.

[0:25:05.9] JM: There are these gigantic companies that do tons of work in the open on github. You have Facebook, you have Microsoft, you have Netflix. How did the open source work styles vary between these different companies?

[0:25:22.1] DA: I don't really feel qualified to answer that, because again, I'm not consumed in any big projects from anyone else. I think the work styles differ a lot, even inside of Facebook, like different teams at open source stuff usually have different priorities. Maybe something internally is on fire and they literally need to spend half a year fixing those internal cases, which might not matter as much the open source community, because they just haven't bumped into those limitations yet.

For example, that happened with Flow, where the external perception is that sometimes Flow doesn't pay enough attention to the needs of the open source community and they started to turn that around with the recent releases. The reason is not because they don't care, but rather internally, or the number of files using Flow is growing so much faster than the performance improvements they could make. They were literally spent most of the year just making Flow scale to the Facebook codebase size in a single repository.

I think the differences in which things are a priority for a team right now determine the style of the communication and the way project is governed and the way the changes are made. Some projects are primarily operated from github and then get synced into Facebook internally. That's how React works; we work on github first.

Some other projects like React Native, they are tied into our internal testing infrastructure and it would be pretty difficult, or maybe even not make as much sense to move it to github first model, because you don't want to lend something that passes the CI tests, but breaks a lot of Facebook products, so it's a balance. React Native has a much larger surface area than React, so it has many more components. It's more difficult to make it github first.

They are also working on reducing the surface area and extraction packages entry into repositories managed by different community members. There is the ongoing lean core effort that tries to do that. I think the surface area of the project can really determine the way it is run. Then there's also the governance. There are very different governance structures. In some cases, there is a small team that makes the decisions after – maybe there is an RFC process we have in React where even when we want to make API changes, we first write up an RFC explaining the motivation and then we get feedback and we might change the proposal.

The governance itself can differ. In some cases, it's a technical committee that is composed from different companies. In some cases it's a single company. The rules for joining a committee like this can also be different. I think a lot depends on the project maturity and also how much of it is in the research phase.

[0:28:48.3] JM: React is undergoing a rearchitecture right now. What have you learned about refactoring large open source projects?

[0:28:59.8] DA: To give some context, I wouldn't say we're currently going through rearchitecture. We actually finished that about a year ago. We are building on top of the new architecture and we are taking advantage of those changes. About a year ago, we basically rewrote React from scratch, at least the majority of React. There wasn't code that stayed the same, but the main engine it was completely rewritten from scratch.

I think there is this common wisdom. I think there is this article by Joel Spolsky from Joel On Software Blog about how rewrites are always a terrible idea and they never work. I think we challenge that wisdom a little bit, but it's challenging to pull it off. I think the reasons our rewrite worked – there were a few different reasons. One of them is we had a pretty clear idea about the things that we wanted to avoid.

We got some new constraints. In particular, we wanted to make rendering interruptible. We wanted it to be possible that you start – that React started some work, starts rendering a large component tree and then there is – maybe there is some user event. Then React can pause that work, update in response to user event that is more important and then rebase and continue working on the large update that's unrelated to the user event.

This was a pretty fundamental change in constraints from the old React and it was very hard to retrofit this into existing code. Sebastian came up with this React fiber project, which is a reimplementation of React around a different algorithm that enables this concurrent rendering when there are multiple updates that are being handled at the same time, potentially the different priorities.

At first, it was very – when he just started, he started sending PRs that – like the pull requests that add first tests and the tests looked like a component that just renders a button and doesn't do anything else and like a counter. At the time, it seemed there is such a long way to go that we'll never get there. It just seems an impossible project.

Then Sophie Alpert started setting up some infrastructure. We realized that we have a lot of tests. Some of these tests were unit tests, where the unit is a particular internal module and some of those tests were more integration tests from React's point of view. Not integration where you load things in the browser, but they were using public API. We realized that we can only really carry over those tests to the new implementation if they test the public API, because all the tests for internal modules are now useless, because we are going to throw away those internal modules and write different ones.

What we did was we had some effort to convert and the Romanian tests that used the private APIs to rephrase those tests in terms of public API and just observable behavior to the user. We spent some time on that and the community really helped. We had a github issue with check boxes, where different people would take a different file and rewrite the test. With that test coverage, what we ended up doing was we took all our existent tests. The majority of those were now written against the public API, and we ran the test suit against both implementations.

We don't do branch development. We did not have a long-running branch or anything. We had a parallel implementation checked into the same repository that was a skeleton at first, and then it started growing mid. We ran both implementations against the same test suit, but of course, the new implementation would fail 80% of the tests. What Sophie did was she made a file that was a list of all paths and tests in the new implementation, which was pretty short at first.

We made the continuous – the CI, the continuous integration server, we made it fail if the list of currently passing tests within the implementation is different from that file. We also had a command that updated their file. The idea there was that you could pick any test from that file, make it pass with a new implementation by implementing the missing features and then rerun the script that would update the list of passing tests, and that will be part of your change.

You could see in each individual change, you could see oh, these are the tests that it fixes. Once you get that merged, when I work in the next fix, you know that it will not regress the old tests, because now those tests are not failing any more. They are not in the file. The CI would fail if you regress on those.

Then our manager, Tom Occhino, so he made a page that had a graph of the number of passing tests with each commit. We had a script that updates the number. I don't know if that website is still alive. Probably not, but you could go to isfiberreadyyet.com and it would have this graph of all the tests – how many tests a person out of how many total we have. It was gradually growing with every commit. I think just opening that page and saying oh, we are one step closer, has had a huge motivating impact on the team. We just kept fixing those.

We also needed to make sure that the unit test is still, even though we have thousands of them, there are still just an approximation of real code. The other thing we did early on was we set it up so that at Facebook, you could opt in to a special flag that would serve you this new implementation instead of the old one. At first, it was completely broken. If you enabled that flag for yourself, none of the website would work. I think the first big step was when somebody implemented handling of CSS classes, so none of the events would work, but it would just set the correct CSS class names for your components. Then suddenly, you could see all the markup jumping into shape and actually looking like a real thing. I think that was also very motivating.

As soon as it was somewhat usable, we opted in our team into that feature flag, so we would all get this new version that is half broken, and we would constantly switch back and forth, because the messenger was not working, all like the task tool. That eventually got pretty stable and we just switched to using the new implementation just for our team full-time. At some point, we decided that okay, it's stable enough that we can enable it for people who participate in the internal Facebook ReactJS group, and so we enable it for 50% of them and started getting more bug reports.

Eventually, it got stable enough that we enabled it for 1% of real users. Again, that caused some bugs and we fixed them. Then we just shipped it to 100% and switched it over to the new one. That's when we knew that it's ready to ship in the open source as well.

[0:37:20.3] JM: All right, a few more high-level questions. Where do you see React in five years?

[0:37:25.7] DA: It's very hard to say, because some of the biggest ideas are completely unforeseen. As an example, we recently released the first stable version of the hooks API. That is a new component API, that it lets you use state and other React features from function components without having to write a class. It solved a bunch of issues that we've been thinking about for the past five years, but Sebastian just came up with it a few months ago and we couldn't predict that that would be with the –

We knew that we wanted to have some functional API, but none of them really looked right. I don't know, he didn't have that idea based on some other ideas that he saw from Dominique and from some people in the community and synthesize them. Maybe this change wouldn't have happened.

I think for a lot of those really impactful ideas, you don't really – you don't really see them in advance. They just sneak up on you when you try to solve other problems. In broad strokes, I think there is a few big missing parts in React that's the – the list of F code names that I referred to earlier, that's our longer-term projects. I think they capture that a little bit.

I'll just say about a few different directions. One of them is the data fetching. The current solutions are not integrated with React itself, which makes sense if you think of React as a view library, but we think of it as a user interface library. In a way, data fetching, or at least being able to wait for something to happen and then present that to the user is a UI concern. Maybe the exact mechanics or not, but the way you reflect that in UI, it is a UI concern.

Some features that we demoed in – so there is a post called beyond sneak peek, beyond React 16 on the React blog that contains a presentation, a talk that I did at JSCon [inaudible 0:39:43.7] that goes into features like suspense that's related to data fetching, and time slicing that's related to being able to render work without blocking the browser. I hope that in five years, those things will definitely be stable. Actually hope they'll be stable this year, but we'll see.

That's part of the picture, just making data fetch much more natural and easier to integrate with React components, while not compromising and actually improving the user experience. I think that is part of a broader topic, which is as I mentioned, we don't just try to find a solution to a problem, because like any problem has a lot of solutions.

We try to find some idiomatic solution that is not a hack, but something that you can solve once and then it stays solved. I think for things like data fetching and doing updates without blocking the browser and figuring out the best tradeoff between server and client-side rendering and perhaps some hybrid solution with streaming, so that you don't have that gap between when the page is just HTML and unresponsive and when the JavaScript loads. Finding some balance there that is have works great by default is a big motivation for the ongoing streaming server render, the new thing that Sebastian is working on.

Even those are not – just scratching the surface of what React would do. There are things like handling gestures, providing better primitives for handling gestures, handling animations, creating UIs that work great on all kinds of devices, including touch devices, tablets, desktop, mobile and finding what's common – maybe some abstractions could be common between React web and React Native, what are those abstractions.

I think there is better abstractions for animations of course, because animations are still not very nice to work with for – from React. They're not very easy to work with. Finding some good solutions there would be nice. Anything that really relates to making better UIs by default, I think that's what we try to do with React.

[0:42:12.5] JM: Okay, last question. You've spent some time reading classics like Code Complete, Joel On Software, you have watched Bret Victor talks, but you also spend plenty of time on the internet and the internet is all about new things. How do you know when to respect your elders and when to ignore best practices?

[0:42:35.8] DA: I try to listen to what Sebastian says. I think he has a very good – he has a very good talent for picking the right thing there, the right balance between those things. The way he does it is interesting. I think he is very good at reasoning from first principles. What he does is he might pick a certain best practice, or something that we've accepted, either in React itself, or

in another library, or in UI development in general. He goes back to the – he goes back further than I see most people go back.

He takes that solution and he's like, “Wait. Why is it done this way? Why do we do it this way?” He goes a step back, okay, this is the problem it was supposed to solve, so this is why the solution looks this way. Then he goes he goes further back and he's like, “Wait. Why is this even a problem? How did we arrive at that problem?”

Because he combines some knowledge from both just low-level understanding of how a computer works, both in terms of CPU and graphics and memory and network and how does the browser work and why does the browser work this way? Or how does a native platform work, or how does language work? I think he just peels back at these layers and he sees that okay, this is a best practice because of a certain set of constraints. What if those constraints were different, or why are the constraints structured this way? Or can we add a different constraint that would alter the path that we're taken?

I think that's the thinking that helps him arrive at these solutions that I don't see many other people managed to arrive at. I think that's my that's my answer is you just peel back the layers behind those best practices and try to trace them back to the first principles. Then maybe the path from the first principles will be different.

[0:44:53.0] JM: Okay. Well Dan, it's been really great talking to you. I appreciate you coming on Software Engineering Daily and talking so much about React and Facebook and your own personal philosophies.

[0:45:02.8] DA: Thank you. It's been fun.

[END]

