EPISODE 639

[INTRODUCTION]

[0:00:00.3] JM: React Native allows developers to reuse front-end code between mobile platforms. A user interface component written in React Native can be used in both iOS and Android code bases. Since React Native allows for code reuse, this can save time for developers, in contrast to a model where completely separate teams have to recreate front-end logic for iOS and android.

React Native was created at Facebook. Facebook itself uses React Native for mobile development and contributes heavily to the open source React Native repository. In 2016, Airbnb started using React Native in a significant portion of their mobile codebase. Over the next two years, Airbnb saw the advantages and the disadvantages of adopting the cross-platform JavaScript-based system. After those two years, the engineering management team at Airbnb came to the conclusion to stop using React Native.

Gabriel Peal is an engineer at Airbnb who was part of the decision to move off of React Native. Gabriel wrote a blog post giving the backstory for React Native at Airbnb and he joins the show to give more detail on the decision. There's a lot in this episode. It was a really good one as far as I'm concerned, and it really gives a lot of color to the advantages and the disadvantages of React Native. Just to be clear, this is not a show that's against React Native. Airbnb has a very specific use case and it's a gigantic company, so it's not to say that React Native is something you should not use. In fact, we've done many shows about the advantages of React Native and I think it's a incredible platform. It's a great show and it gives both sides of the story.

Before we get started, we're hiring a creative operations lead. If you are an excellent communicator, please check out our job posting for creative operations at softwareengineeringdaily.com/jobs. This is a great job for someone who just graduated a coding bootcamp, or someone with a background in the arts who is making their way into technology. If you want to be creative and you want to learn more about engineering and you have an excellent work ethic, check it out at softwareengineeringdaily.com/jobs.

[SPONSOR MESSAGE]

[0:02:27.6] JM: Cloud computing can get expensive. If you're spending too much money on your cloud infrastructure, check out DoIT International. DoIT International helps startups optimize the cost of their workloads across Google Cloud and AWS, so that the startups can spend more time building their new software and less time reducing their cost.

DoIT International helps clients optimize their costs, and if your cloud bill is over $10,000 per month, you can get a free cost optimization assessment by going to doit-intl.com/sedaily. That's D-O-I-T-I-N-T-L.com/sedaily. This assessment will show you how you can save money on your cloud, and DoIT International is offering it to our listeners for free. They normally charge $5,000 for this assessment, but DoIT International is offering it free to listeners of the show with more than $10,000 in monthly spend.

If you don't know whether or not you're spending $10,000 if your company is that big, there's a good chance you're spending $10,000, so maybe go ask somebody else in the finance department.

DoIT International is a company that's made up of experts in cloud engineering and optimization. They can help you run your infrastructure more efficiently by helping you use commitments, spot instances, right sizing and unique purchasing techniques. This to me sounds extremely domain-specific, so it makes sense to me from that perspective to hire a team of people who can help you figure out how to implement these techniques. DoIT International can help you write more efficient code, they can help you build more efficient infrastructure. They also have their own custom software that they've written, which is a complete cost optimization platform for Google Cloud, and that's available at reoptimize.io is a free service, if you want to check out what DoIT International is capable of building.

DoIT International are experts in cloud cost optimization. If you're spending more than $10,000, you can get a free assessment by going to doit-intl.com/sedaily and see how much money you can save on your cloud deployment.

[INTERVIEW]

[0:04:51.1] JM: Gabriel Peal is a software engineer at Airbnb and he's a writer of a blog post about React Native at Airbnb. Gabriel, welcome back to Software Engineering Daily.

[0:05:01.0] GP: Thank you so much for having me.

[0:05:02.3] JM: Airbnb has been using React Native for developing mobile apps for the past two years. I want to start from the beginning and get through some of the story of React Native at Airbnb. How were mobile apps developed before React Native was at Airbnb?

[0:05:20.9] GP: Yeah. This is an interesting one. When we go back to the earliest days of mobile at Airbnb, they really started in around the 2012 timeframe. At that point, mobile was pretty young, we were just a website, and over time we realized the importance of having a mobile app. Not just for booking a listing, but also for travelers, because often when you travel, all you have is your phone. It was really important for us to have a really good mobile experience.

2012, 2013 is when we started to build the early stages of our mobile apps, which then grew into about a 15 person team on each platform; Android and iOS in the middle of 2016. Now at that point, we were starting to see a huge influx of mobile traffic. Our mobile usage was going through the roof and it went from this tiny, tiny fraction of the business as something that was extremely important. Then teams were encouraged to make sure that their features worked on mobile, on Android and iOS. With 15 people on each platform, we literally just didn't have enough people to build what we needed.

[0:06:28.0] JM: Well, that team structure issue, that was the whole idea behind React Native is that React Native would give you reusable components that would allow for some communication centralization within different teams. You can design a component and make it portable between Android and iOS and web, that was the goal of React Native, if I understand it correctly. I'm sure when React Native came out it was like, “Oh, that's our problem.”

[0:06:59.7] GP: Yes. I guess that there are two similar, but different mantras that people take when they think about React. There is a learn once right everywhere mantra, where you learn the React paradigm, you learn JavaScript, JSX and the ecosystem and then you can write on any platform. Then there's write once, run anywhere, where you literally write the code once and it runs on multiple platforms.

In this particular scope of React Native, it's actually a little bit of both and it's important to understand why they're different and understand which one you're trying to glean the most value out of. It started from the learn once write anywhere paradigm, because at Airbnb at that point early on, we had about 15 Android and 15 iOS engineers, but probably well over a hundred people in the company who knew React. Our entire website is React, is very successful for us rewriting our website in React. We've also built a lot of infrastructure and libraries and are highly leveraging the ecosystem around React.

We saw that as an opportunity to build on our existing native infrastructure, but then allow people who know React to contribute to mobile. That would be their learnings from web on mobile. That being said, over time it becomes clear that because React Native is in fact React, it's not a fork of React, it's not a clone of React, it's actually React running on your phone. Because of that, you literally can share a lot of code, you can leverage a lot of the same libraries. We use things like Redux and a number of other industry standard libraries in the React community.

Although there's some organizational challenges and actually starting to share code between web and mobile, we saw the potential to do that either by directly sharing code, or by having the same person write the feature once on web and once in mobile, because they could reuse a lot of their thought processes.

[0:08:58.1] JM: When you started using React Native, were you just experimenting with it a little bit, or did you make a really decisive decision to let's just do it, let's go all-in on React Native.

[0:09:13.0] GP: It depends who you ask. This is actually one of the more contentious organizational challenges we had with React Native. It was started as essentially an experiment. I think it's really important in the world of engineering to treat this everything like an engineering problem. It has pros and it has cons and it's really important to understand the full set, or as large of a set of pros and cons as you can in order to mitigate risk and help have a smooth and not rocky rollout. That's really, really important.

It started like an experiment. There were essentially two engineers; Leland Richardson and Spike Brehm, who were very experienced react engineers and they worked in partnership with some people on mobile like myself and a few others to build out a product on mobile. We both thought it was a reservation alterations. That was the first thing we ever launched with React Native. We started building it out in July 2016 and then it launched in October. In that process, we had to build out a bunch of things like experimentation, internationalization, deep links, navigation, our design language and a number of other things.

Then things started to change really quickly around, right around that same time, because in November of 2016, we launched experiences, which so you can book something to do in a city and in addition to a place to stay through Airbnb. This particular team had lots of web resources and not enough mobile resources to actually build out what they needed to before launch. That became this pivotal moment for React Native, because that team in particular just decided to go all in on React Native for their products, prior to us ever launching the first one for a really, really important and highly visible launch.

It was an opportunity that had – it could have got a number of different ways. It certainly wasn't painless. I think it was hectic getting that out the door, there was a lot of work to be done, a lot of long hours, but at the end of the day it did launch. It launched on time. Did not launch bug-free, but it did launched on time and they would not have been able to finish building out their feature set without pulling engineers off of other teams in order to launch that time. It became this interesting case where one team just for – their hand got forced into this direction and it ended up really dramatically accelerating the React Native adoption very early.

[0:11:44.7] JM: Okay, so let me see if I understand correctly. React Native comes out, different teams on Airbnb mobile development said, "Yeah. Let's tinker around with this. Let's experiment with it, let's do a few things, internationalization, etc. Then experiences gets announced as a feature that we're going to build and the experiences team makes a very rational decision, we've got a lot of web resources, we've got fewer mobile resources. Well, let's see if we can go all-in with React Native. Let's see what happens." They went all-in with React Native and they did ship on time. They had to pull people off of other teams, but it sounds like it did get shipped okay. Do I understand things correctly so far?

[0:12:29.5] GP: It launched. With every launch, there are – few launches are perfect and smooth and bug-free, and this was not one of those just like others, but it went out the door, people were able to use it, people were able to book on it. In a sense, it did allow them to get to their goal.

[0:12:46.3] JM: I sense that there were some – it sounds like there were some foundational things that as experiences was being built, or as it was being released, you were like, “Oh, no. There's something wrong here.”

[0:12:58.2] GP: I think it's really important to understand exactly, yeah, break it down a little bit further. React Native is when you add React Native to an existing code base, especially if it's a larger one with a lot of infrastructure of its own, as ours was and still is, it's really important to think about what that interaction is going to look like and what institutional infrastructure that you have to either recreate, rebuild or bridge. In this particular situation, it was like a was a cat-and-mouse game, where they’re just – in order to build a screen, you want to be able to leverage our networking stack, you want to be able – it was critical that we run experimentations, all the internationalization has to go through an internationalization pipeline. There are no other options. If we want to actually ship a product, we have to do that work upfront.

There was just a huge amount of work that had to be done to get from 0 to 1, in the case of React Native. I mean, it was not the thing we could just tack on top and just write one screen in isolation and ship it. It just simply – that would never have been a good experience and it would have hamstrung the teams that would have chosen to use it.

I think in this particular case, because everything was happening so quickly, I think there was a combination of things that were made difficult specifically because of React Native, but also simply because we were tackling on a huge new platform and there was just an incredible amount of work that had to be done, and it was just a matter of hours in some cases.

[0:14:27.2] JM: You and I were talking before the show about how, now the main thing you're focused on is Android infrastructure and you said something like 80,000 lines of code in the Android app.

[0:14:36.5] GP: 800,000.

[0:14:37.3] JM: 800,000. Okay, right. I'm sure there's something proportional on the iOS side of things, and it's like, if you imagine the amount of infrastructure that has been built around managing an 800,000 line Android app, and I'm sure it wasn't 800,000 lines back when you were launching experiences, but just probably something still tremendous, you think about introducing the necessary build tools, or whatever other supporting infrastructure you need to put into a mobile app to make React Native work, not to mention all of the organizational reconfiguration that you need to do in order to ship a feature with this new hybrid platform. You're potentially incurring a lot of strange – or not strange, but fresh types of technical debt.

[0:15:28.2] GP: Yeah. Then those things are going to need to be forever maintained as well. When your company comes out with experimentation platform V2, or API V3, or starts rolling out graph, moving from rest to graph QL, somebody has to always be there to keep React Native up to speed, or it's just going to fall further and further behind.

[0:15:46.4] JM: When experience is launched and you were – I guess, you were having this realization that you did incur this fresh technical debt, what was going through your head? Were you thinking I need to bring this up, or we need to reconsider React Native, or what were you thinking? We're going to have to push – we're going to have to bring in a lot of support to make React Native work. Were you gung ho for React Native at that point, are you starting to feel nervous about it?

[0:16:15.9] GP: It's important to consider, and a particular question like this that one – just one thing I want to call out about React Native is that it's a incredibly polarizing topic, and there's no way around that. A lot of companies are choosing to use it and not choosing to use it, but almost in every case there are people within an organization who are very pro React Native and/or cross-platform in general and very against it. The answer that you hear is going to vary tremendously based on who you ask.

From my personal perspective, I understood why React Native was the cross-platform framework that we chose. It made a lot of sense to be able to leverage our existing infrastructure and talent here, because we have a lot of it. That being said, I think experiences betting the farm on React Native was a huge risk. I think that there were certain aspects in which they had to sacrifice the product quality as a result of jumping to React Native and going all in on it so quickly.

I'm someone who I care deeply about product, so things like it has to be smooth, it has to be responsive, it has to feel amazing to use. We can achieve that with Native, and so I would be worried with React Native if I felt like we couldn't actually achieve that. That being said with React Native, surprisingly we were able to get closer to that bar than most people would think as possible.

I work closely with a couple of other engineers and we got things like shared element transitions working between Native and React Native screens, and an API that was canonical in React and also worked in iOS. Things like this, I think it goes to show that if you put in enough effort, you can actually overcome most of the technical challenges that may be associated with React Native. One of the things that we discovered is that understanding how much effort is going to be needed is something that's very difficult for a number of reasons.

One of the big reasons is that React Native inherently is not just one platform. It's actually three platforms baked into one. If you're working on React Native, most of the time you're living in JavaScript in React land, but you can't forget that it's running on Android and iOS under the hood, and there are times when that the native implementations peek through, or you have to dig down and understand the nuances of each platform in order to do something appropriately.

That creates a challenge, where if you're working in one platform, you can learn that platform through and through. You can become an expert at it and you can understand exactly where to poke and where to prod and where to look to make it do – to bend it to your will essentially. I've yet to meet an engineer who can do that effectively on three platforms at the same time.

Getting to some of these last 5% or 10% sometimes requires some explicit and intricate knowledge of a platform, and being able to do that effectively in React Native is a really hard problem and I have yet to see an easy answer to that, because at the end of the day it requires understanding three platforms.

[SPONSOR MESSAGE]

[0:19:28.4] JM: We are all looking for a dream job. Thanks to the internet, it's gotten easier to get matched up with an ideal job. Vettery is an online hiring marketplace that connects highly qualified jobseekers with inspiring companies. Once you have been vetted and accepted to Vettery, companies reach out directly to you, because they know you are a high-quality candidate. The Vettery matching algorithm shows off your profile to hiring managers looking for someone with your skills, your experience and your preferences. Because you've been vetted and you're a highly qualified candidate, you should be able to find something that suits your preferences.

To check out Vettery and apply, go to vettery.com/sedaily for more information. Vettery is completely free for job seekers. There's 4,000 growing companies, from startups to large corporations that have partnered with Vettery and will have a direct connection to access your profile. There are full-time jobs, contract roles, remote job listings with a variety of technical roles in all industries, and you can sign up on vettery.com/sedaily and get a $500 bonus if you accept a job through Vettery.

Get started on your new career path today. Get connected to a network of 4,000 companies and get vetted and accepted to Vettery by going to vettery.com/sedaily. Thank you to Vettery for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:21:12.0] JM: Okay, so what you're alluding to is the fact that in React Native, if you're just doing fairly basic app-based interactions, then React Native is mostly going to take care of it for you and you can stay in JavaScript land and everything, from what I have heard, React Native is pretty great for that, if you have fairly basic interactions. Once you get to really complex interactions, or something that needs to be highly performant, then you might need to dip down into the native code. React Native is in this JavaScript land with this JavaScript bridge and you might need to actually write Objective-C code, or Swift code on iOS, or write Java code on Android.

Therefore, if you need to duck down on a regular basis, I can see it being quite troublesome, because – help me understand, why wouldn't it work just to have – you've got a bunch of React Native engineers and maybe they do most of their development in JavaScript, and then occasionally they have to flag over the iOS expert to come over and help them with their low-level feature. Do you have to dip down into iOS code, into Swift code, or Objective-C code more frequently than you would have hoped, is that what happened?

[0:22:24.6] JM: I think the answer is a little bit complicated. I think, to answer the first part of that question about React Native engineers sometimes needing to pull over the Android or iOS engineer for help. There's two parts of that. One is understanding where exactly the problem is. In React Native, there's a lot of moving pieces; there's the react platform, there's the React Native stuff in JavaScript, there's the bridges to Native, there's the Native implementation in Java and Objective-C, and then there are native libraries such as yoga that handles the layout and a few other things as well, as well as the JavaScript runtime itself.

There's a lot more black boxes in the React Native world. If it were simple to always pinpoint exactly why something is behaving the way that it is, I think you can make the case that you could do that. We encountered so many situations where something would happen and it's just almost unfathomable how you would even go about debugging it, because it could be coming from so many different places.

You could certainly – you have one debugging environment on the JavaScript side and you have another debugging environment on the Android iOS side. Then God forbid, if your issue is somehow layout related, or you're having layout performance issues in yoga, figuring out how to debug that can be a nightmare. These are situations that you don't really encounter on Native very frequently.

[0:23:50.9] JM: Okay. Let's talk a little bit about that tooling that you developed. As React Native became something that was a central part of both the Android and the iOS apps, what tooling did you have to include to help with things like debugging, to help with things like build time, I don't know what other issues you have to solve at the library, or tooling level, what kinds of standardization did you have to put in place?

[0:24:19.6] GP: Yeah. We had a fairly complex build, a Gradle file and iOS build file on each platform that knew how to use watchman to watch for changes in JavaScript and build the bundle locally. We don't check in the JavaScript bundle for a React Native into our repo, because it's fairly large, it's split up into split bundles and it changes frequently, so it would just be a lot of stuff to commit to the repo. It gets generated on the fly, on each person's computer whenever it changes.

We had to build an infrastructure on that and make sure that the caching works for that. There are some complications on how to handle React Native libraries that have native components. On iOS, it's a little bit complicated, because we use CocoaPods on iOS and then you have to do some weird sim linking, where you sim link some CocoaPod sources into node modules, or vice versa. Then on the Android side, it's a little bit more frustrating for us at least, because, or at least a lot of the Android community is used to using Gradle. They use Gradle and they publish most libraries to maven, so you can simply add one line of code in your Gradle file and it generally works quite well.

A lot of React Native libraries require you to link this source from node modules. You install your node module, NPM install whatever, and then you have to in your Gradle file link to that source there, which is fairly atypical and non-standard if it goes against everything else that you do. We we actually wound up having an internal maven repo, or we had to get libraries and then republish them internally and it caused – little things like that caused all kinds of pain and overhead and random places that you wouldn't really expect.

[0:26:03.7] JM: What were the organizational consequences of these changes to build time and debugging? How did it affect the overall product development, and maybe you could give us a point in time here, where are we in time right now? After experience is launched, was React

Native continuing to be first-class citizen, or was this one you were starting to have doubts? Give me a little more context.

[0:26:29.7] GP: Yeah, I think it's important to just call out how big of a role React Native played in our mobile development. This is something that I think there was a big misunderstanding of among the community. At the point where we launched experiences, because it was a big important launch for us, most of experiences was React Native and there wasn't much else in React Native at that time. That amounted to roughly 20% of our engineering work at that point in time.

Then after that, some of the experiences were stabilized. It wasn't quite as intense there, but then some other teams ramped up on it. Over time, we saw a fairly stable cohort of people that amounted to roughly 15% of mobile engineers that were working on React Native at any given time. It never really grew so much more than that. Somehow, whether it be us talking about React Native externally, I think the misinterpretation of us being a 100% React Native or moving from Native to React Native, somehow became the thought in the industry. In fact, it's something we continued in parallel. It would account – again, accounted for about 15%.

On the infrastructure side, we had – it was myself and Leland Richardson working full-time on the React Native infrastructure for 2017. That's about the same number of Engineers we had each on Android and iOS that were purely focused on infrastructure. About six total, plus about two or three more who work on native builds and CI. It was never much more than that. Medium-sized effort overall, but it's certainly not a 100% over that.

[0:28:09.7] JM: How did that affect the organization? How are you feeling organizationally about the impacts of React Native?

[0:28:18.8] GP: I think it depends who you ask. For teams that decided to not use React Native, I think there was actually very little impact. Does things, like I mentioned like building the JavaScript asset locally, so that added sometimes a minute, or maybe two or three minutes, just a clean build every once in a while when necessary. I think there are things we could have done to make that a little bit faster.

That was probably the – that was the majority of the extent to which they had to deal with React Native. Then again, there were other teams who basically invested a $100 in React Native. From when they started working on it until when we worked with them to work on moving away from it, they didn't have to hire a single Android or iOS engineer. That being said, it wasn't – that was not to say they never had any work in Native, Android or iOS.

Many of those engineers had some experience on one platform or the other, but also part of the reason why Leland and I worked full-time on React Native infrastructure was that we provided a significant amount of support. We made ourselves available to other teams to help them when they were confused, when they were stuck, or when they needed to have some native work done.

This can be a challenge organizationally, because people – different teams have different incentives and stakeholders, and it's really important to understand that in a world like this, if you're going to have a team that's ready to write bridges, or help, or debug, or provide mentorship, it's really important to understand that that team – part of that team's incentives need to be aligned such that they're willing to invest that time directly, and almost in a randomized way, because they can come in at any time for other teams that might need it. That understanding between the teams of help is something that needs to be understood by the organization, or otherwise one team is essentially doing charity work for another team, and they'll wind up frustrated, or wind up, in worst case, resentful having to do free work essentially for another team.

[0:30:15.7] JM: Okay. Before we talk about migrating away from React Native, I want to give my caveat as well that I think React Native is an amazing project. I've done, I think 20, or 50, or something shows on React – definitely not 50, but I've done a lot of shows on React Native and I think it's a great product, project. It's really exciting. Every company is not Airbnb, so like you published this blog post about moving off of React Native at Airbnb, that doesn't mean everybody should be off of React Native.

Airbnb is a really big company, you've got 800,000 lines of code. Also, by the way, you're in a very unique business situation, where Airbnb has one of the biggest moats in technology. You've got a huge moat and you – from a business perspective, or from a technology perspective, like I think right now, Airbnb is really in the phase of like, “Let's get our house in order and let's do very straight forward.”

I mean, the business and model is so good that it's like, “We can just do the boring things on the engineering side and the business will work really well.” I'm not saying that's what you were doing here, what you had to do, I think it's, I don't know. You could certainly go that way and make a good business. You don't really have to do anything super fancy on the engineering level, other than scale, which is fancy in and of itself.

[0:31:31.7] GP: I’m not sure I agree with that.

[0:31:33.2] JM: Really. Okay.

[0:31:34.2] GP: Yeah. I mean, the business is doing well and that's always great. I think for us, we are more so than a lot of companies were very design-first. The co-founders of this company are designers. The ability to create really incredible experiences has been really core to the DNA of Airbnb since the beginning. The decision to move away from React Native was not even directly a result of us meeting, or not meeting product goals, or our ability to achieve those. It was simply a matter of practicality.

I mean, I tried to outline it in as much detail as in the blog post series as I could, but I think that there were just some challenges technically and organizationally that such that the benefit that we were getting out of it was no longer worth the investment. Based on the people who are working on it, both myself and Leland on the infrastructure side, as well as the product engineers, I just saw that there was a lot of opportunities to make Native even better.

At the same time, I think React Native was actually going fairly well and the teams that – there were teams that were very – were able to use it very effectively and to their advantage. The overhead of maintaining the infrastructure, both maintaining it on its own, as well as the opportunity cost of those people, then working back in – like contributing their work directly to Native, Android and iOS just wasn't worth it.

Now that there's more resources on Android and iOS again, we're actually seeing some material improvements in both, the things like build times, so many open source libraries. I'm really excited about something that we're working on Android that we're coming up with very soon. In fact, some of the principles of that were inspired by some of the things that work well in React Native.

Again, it's not about us just settling down the easy path, I think is just a matter of the opportunity cost was very high, it's a very polarizing subject, and there wasn't really a path for React Native to ever be more than 15% or 20% of mobile here.

[0:33:31.7] JM: Right. Okay. I think I'll just shut up and ask you this question. When you decided that you were going to move from React Native to sunset React Native and move to purely native mobile platforms, what was your forecast for how that would change the build and the engineering, how that would make you move faster?

[0:33:55.5] GP: Yeah. At the time when we decided this is sunsetted, we had about two engineers were going to infrastructure and again, about 15% of mobile engineers. Leland and I, we sat down and we looked at all the teams that were using it, and then we did some for road mapping to figure out what teams may start using it in the future, what new opportunities are opening up that it may be a good fit for. We didn't see that growth potential.

The thing about cross-platform and both Android iOS, but also building robust infrastructure to do things like code sharing between web and mobile, and then also building infrastructure around code push, which we haven't talked about yet, but would enable over-the-air updating, all of those things are highly leveraged based on its usage. Without significant usage across our mobile products, it no longer justified the investment. It really just came down to the fact that there wasn't a real opportunity for it to become much more than 15% or 20% of mobile.

[0:35:02.1] JM: 15%, 20% of the mobile codebase, you mean?

[0:35:05.2] GP: Mobile codebase, the number of engineers contributing to React Native versus Android or iOS.

[0:35:09.3] JM: Okay. When you decided to do this, or when you proposed it, was that divisive as well, or at that point were people organizationally realizing the cost that were being incurred?

[0:35:23.7] GP: I would say, it's gone about as smoothly as you could have hoped it would. Certainly, like I mentioned there were a few teams that had really effectively leveraged React Native. They had effectively gone all-in on it and resourced their team appropriately with people who knew React Native at the expense of Android and iOS engineers. It took a little bit of additional work and we've provided some additional resources and hands-on mentorship and actually in some cases, direct help to rewrite some of these features in Native. At the end of the day, I think that once the landscape was outlined to them, it really only – it was a fairly understandable decision for most teams, and then for the last few that really depended on it. I think they understood fairly quickly and there hasn't been much pushback since.

[0:36:09.8] JM: By the way, what are the things that you have to duck down into native code to do on Android, or iOS, if you're using React Native? How often do you have to do that?

[0:36:21.9] GP: It depends on what you're building, but some of the things that we bridged were on the infrastructure side, there were things like our networking stack, experimentation, internationalization, deep links, navigation. Then on the product code side, it was things like geo-fencing, things like maps, video views, lotty and a few other things like that.

Some of these cases, some of those cases are wrapping views, but they can be quite complex. Let's take the video and maps example. They're very, very complex. They have a lot of functionality. Sometimes it depends on the hardware, and it can be frustrating sometimes when you start up your testing React Native on iOS, you write it and everything looks good, you go to Android and it doesn't work exactly the same.

Now this is a very frustrating situation, because the whole point of React Native is that it should just work in both platforms. We found very frequently and in the case of React Native maps, that was one of our own libraries that I think we could have done a better job maintaining, but takes an incredible amount of time and energy to do so, and is quite a complex product. When you find that you have a bug on one platform or another on React Native Native library, it's very frustrating and it's very common.

We found that many of these libraries on React Native were written by an engineer, or engineers who were very familiar with one or two of the three platforms, but frequently lagged in in one of them. This is to be expected, right? If your one engineer writing an open-source library, how could you possibly write perfect code across all three platforms? It's just these unicorn engineers simply don't exist in the world.

As a result, you find that there's often significant code quality issues on at least one of the platforms. That's an example of when things good and native and some sometimes, it's those are things we've built in-house and sometimes they're in third-party libraries.

[0:38:11.3] JM: How did the – when you were in the thick of it, when you had, or I guess you still do have large-scale React Native infrastructure, but how does it affect releases?

[0:38:20.1] GP: For the most part, React Native was the pure release process; React Native didn't have a big impact on. We shipped a bundle with the app and we didn't update it during the life of an app and the wild. Towards the end of our React Native lifetime, we were working on some code push infrastructure that would allow us to update that React Native bundle over the year. We actually got fairly far and we had an entire rollout plan that would have had several phases going from hot fixes and cherry-picks, to fixing major issues, to eventually getting to a world where we could feasibly see a world where we have continuous deployment on mobile.

That's an incredible world. It's not possible with native apps, with the review process. It's a beautiful promise, but it would have taken a lot of work to get there, not just technically, but also organizationally. Understanding when you when you do have discrete app releases and then a continuously updating bundle, there are a lot of things you have to consider. You have to safeguard it in a lot of ways with automatic roll backs and minimum versions and things like that to ensure that it has the same native API, to ensure that I mean, you simply – you can't test a new bundle across every app version manually, you can't send that through a QA process, so we had to think about some of those problems ahead of time.

I think the community will continue to work on this problem and it's something that React Native enables, which I think is very cool. We didn't get there. Also on the release process, we had to make sure that we could understand crashes in the wild. JavaScript crashes by themselves, just bubble up as JSC exceptions on Native, and so we had to do a little bit of additional crash end link to make sure we caught the actual crash. We uploaded that to bug snag, which is what we use for crash analytics and then properly make sure that source maps are uploaded as well.

There was a fair amount of effort and we actually – we had to work with bug snag to make sure that they supported JavaScript source maps in addition to native ones, to make sure that that all got fired up and it also introduced a new opportunity for our own infrastructure to fail. I think we had one or two releases early on, where some of our wiring that connects the React Native crash to a normal blog snag crash and is properly source mapped and things like that, that broke a couple times, because there was just a lot of moving pieces there.

We did see more instability there than we did, than we saw in Native. In addition for you Android engineers out there, you may be able to sympathize with this, but we've wound up with just so many random OEM devices that crash randomly, on certain versions of Android, or in certain regions. We're still dealing with this really nasty crash. It only occurs on about six or eight different Samsung devices on a specific version of the software, and we keep buying these devices. We're custom flashing them with these ROMs and we cannot reproduce these crashes that occur so specifically on certain devices.

We've seen a number of these, like not just one or two, but over since we've launched React Native, we've probably accumulated tens or hundreds of thousands of individual crashes from these super obscure native library loading and middle of React Native on Lollipop, at Samsung, or Huawei, they're just mind numbing how to figure out how to fix that.

[SPONSOR MESSAGE]

[0:41:47.7] JM: Test Collab is a modern test management solution which integrates with your current issue manager out of the box. For all development teams, it's necessary to keep software quality in check, but testing is complex. There are various features to tests, there's various configurations and browsers and platforms. How do you solve that problem? That's where Test Collab comes in. Test Collab enables collaboration between software testers and developers. It offers wonderful features like one-click bug reporting while the tester is running the tests, collaboration on test cases, test executions, test automation integration and time tracking.

It also lets you do a detailed test planning, so that you can configure platforms and browsers and configurations or any variables in your application and divide these tasks effortlessly among your team for quality assurance. All of the manual tests run by your team are recorded and saved for analysis automatically, so you'll see how many test cases passed and failed with plenty of QA metrics and intelligent reports, which will help your applications quality.

It's very flexible to use and it fits your development cycle. Check it out on testcollab.com/sedaily. That's T-E-S-T-C-O-L-L-A-B.com/sedaily. Testcollab.com/sedaily.

[INTERVIEW CONTINUED]

[0:43:26.8] JM: We've talked about the debugging process, the tooling – the bugging process is difficult. The amount of tooling you have to include can be difficult, and the fact that you have to write JavaScript in for a lot of that changes the team structure, it changes communication, what are some other subtle problems or changes that you had to make to the process of writing the mobile application due to the fact that you had adopted React Native?

[0:43:55.7] GP: Okay. I'll give you a good example. Take Android, or other couple examples, okay. We have Android. On Android, it does this peculiar thing where well sometimes kill you the process of your app in the background, ask you to save some state in a parse level bundle and then actually restore your app in a new process, to make it feel faster. You can store things like the ID of a product page, and then which you can use to refetch it. It'll also recreate, synthetically recreate a back stack and whatnot, so that even though your app was killed, it feels like it wasn't.

This is really cool and subtle things like this can help improve the time to interactive, it can bring the user back to where they were, it's a good user experience. On the React Native side, we were using Redux to store a state. Now Redux is just a JavaScript object that's floating in space. There's simply no way to reliably persist that in this parse level bundle. We thought about a number of different options, like we could go through that and figure out maybe which parts of it are persistable and only persist those, or marking certain things as persistable, but then you have a react engineer who doesn't really understand Android well enough and knowing the right places to do that is a nightmare.

If you only persist some things and not others, automatically you can actually wind up in a super broken situation, where you've only restored half your state, but it's in this illogical state that makes no sense. Unfortunately, we just had to resort to – we did a little hack to determine if we were in a different process and we just finished all of our React Native activities. We basically just blew away all that behavior. It's really unfortunate and I haven't heard of any good solutions to that. That was one thing.

Another thing that was pretty tricky was figuring out how to handle text inputs and a scrollable screen. One of the things you need to make sure you do, is if you touch a text input that's below the top of the keyboard, you need to scroll that screen to bring the text input into view. This is extremely tricky. iOS makes it a little bit easier. There's one keyboard,, there's one way to handle it and you can more or less make that work on iOS.

Then on Android, you're starting – there's two top-level options. Sometimes, basically you have to configure an activity in Android, you tell it what to do when the keyboard pops up. You can tell it to either do nothing, or you tell it to resize the window to make room for the keyboard. I did this over a year ago now, so trying to remember the specifics, but I had probably spent two straight weeks dealing with understanding the window on Android versus iOS and understanding when the keyboard is up and taking an account that on Android, you can have any number of different keyboards, you can have one that's split on the left and right side of the screen, you can have one that's floating, and there's so many different configurations for the way keyboards can work on Android, plus different heights of auto-correction rows and additional padding and things like that.

Something simple like a screen with form, like a login and password, or maybe some other form, and having a text input field and making sure that it does the right thing with the keyboard, all of a sudden takes two weeks when you didn't expect that to happen.

[0:47:06.5] JM: What's the roadmap for Sunsetting React Native? What's your plan for transitioning teams off of this technology?

[0:47:15.7] GP: Be prior to announcing it's the rest of the company. We sat down with the engineering manager of every single team that's working on React Native, to you understand what the impact of it was. Then we have a spreadsheet now internally where we have all the React Native projects and their owners and their the process of moving away from it. I can speak more on the Android side, because that's what I'm more familiar with, but I think the teams have been very, very cooperative and I'm very thankful for that. Such that, they all are making sure that they have resources on Android and iOS to move away from it.

At a high-level, we agreed to essentially maintain support for React Native through 2018 and maybe a little bit into 2019. Basically, we're saying, "Hey, we threw this at you really quickly. We don't want to randomize your current roadmap, and so we're going to make sure it doesn't break for at least a year or a year and a half. After that, you should really start to move away from it, because we're not going to be putting as much effort into maintaining the infrastructure." That was one side of things.

The other side on the Android side, we've used this opportunity of moving away from React Native, but also doing things like adopting Kotlin, which we've done very well this year. We've gone from about 0 to 80% of new code in Kotlin in 2018 alone. We've used this opportunity to take some of the best aspects of the functional reactive nature of React and we've built this really nice Android Kotlin library or framework that leverages some of the – basically a lot of the common things we already do in screens, and it wraps up some of these patterns into a really nice framework that's both fast to develop in, but also it'll feel familiar for people who are used to React Native, because it has some similar concepts.

[0:48:55.4] JM: Do you have any advice for people regarding React Native? Who should use React Native? Who should not use React Native? I don't mean to make you prescriptive, maybe you just want to talk about specific strategic decisions, but can you help people vet this technology?

[0:49:14.3] GP: Yeah. This is the golden question, should I use React Native or not? I refuse to give a specific, you should use React Native, or you shouldn't, but this is what I would say; first of all, doing React Native does not preclude you from ever having to do Native, Android or iOS. There will, unless you hire people who explicitly have to do that, just be aware that if you do React Native, you will need to jump into Android or iOS fairly frequently depending on what you're doing.

The second one is that when React Native works, it is amazing. I saw we had a couple teams that had really good experiences with React Native and the productivity and the speed with which they were able to move was simply off the charts. Like between hot module reloading, which actually works reliably, you write a line of code and it shows up on Android and iOS in one or two seconds, I think that's really incredible. What you wind up with is these, what I call land mines.

Everything is going swimmingly well, and then you hit a little landmine. I'll give you an example of one. We had one instance where sometimes on certain phones, even in particular the pixel was particularly notable for this, randomly we didn't feel we had made any significant changes. out of 10 times or so, all React Native screens would render white. They would never render, and we didn't know why. Everything seemed fine. It was initializing, it was hitting the JavaScript, but it just simply wouldn't show up on the screen.

We were pulling our hair out trying to reproduce it reliably, figure out what is going on. After I would say a solid week of multiple engineers going heads down, trying to figure out what was going on, we discovered that we had removed the initialization of fresco. Fresco is React Native's image loading library to load images from a network. At Airbnb, we use Glide, which is basically it's a similar library, but instead of using theirs, we've always used Glide, and so we just wrapped the image tag in React Native with our own image view.

The React Native library has fresco out of the box, and so we had that included. All we did was remove the initialization of that and it randomly caused on certain phones some screens to never render React Native. To this day, we have no – absolutely no idea what the connection was. It's been multiple engineers all of a sudden had to spend an entire week trying to figure out how to make this work again. The These are the release blocker and screens were simply not rendering, so this is really, really bad.

Other issues like the keyboard thing that I mentioned, I think you're like, “Oh, it's just a simple form,” and you write the components for the form in an hour and then you spend two weeks trying to make it scroll above the keyboard. Things like that were really, really difficult for us, because it makes it really hard to forecast how long things take, or you end up giving up and even though it's technically possible to make it do what you want, the amount of effort it is to figure out how to solve that landmine is not worth the investment.

I don't have an answer for this. I think teams, like teams just need to be aware that this is the case. Teams often will do a prototype and their prototype will go really well and they'll be like, “Great. React Native is the best thing ever.” Then as soon as they try to take it that last 10% or 15% to production ready is when they encounter all these problems.

The other thing that I'd say is extremely, extremely important is that React Native requires significant and continuous investment in infrastructure. I’ll give you an example. We maintain a fork of React Native at Airbnb. We don't want to do this, but I'll walk you through the practicality of the situation. We really, really wanted to get our screens to have better accessibility. This is just one of many examples, but React Native accessibility support is lacking. This is another case where you think everything is great, you're like, “Oh, I'll just make it accessible. All the APIs exist on Android and iOS, but they're not plumbed all the way through the React Native [inaudible 0:53:09.2] and view system, and so you have to end up building it yourself.”

We went ahead, we built it ourselves, but that change had to go into React Native core. We could of course – React Native being an open source platform, we could have gone to the Facebook repo, we could have put up a pull request, got it approved, merged it, but then we would have to wait for weeks for it to go into the next release and then run that update locally. The turnaround time for that would probably be six plus weeks, plus many, many additional hours. Sometimes we do that in parallel to also cherry-picking that onto our own fork.
Now it happens. We have some commits on our fork. In fact, over the two years that number grew to about 50, 50 individual commits. Again, we don't love this situation. We would love to just get it back React Native, but sometimes when you just need to get your work done, you

have to do things a certain way. Now every time there's a new React Native upgrade, which is once a month, we have to manually cherry-pick 50 commits back on top of the tree.

Of course, React Native is moving very quickly, it's progressing, and so very, very frequently we encounter merge conflicts, or just sometimes they're small, but sometimes the entire files changed. The entire file is missing, or the entire library is totally uprooted from underneath you. You wind up in this situation where you basically get stuck. Around React Native, at 46 now I think. I'm fairly certain that we'll never be able to upgrade React Native ever again at Airbnb at this point.

This is not a good situation and it's something that it's very, very easily easy to accidentally find yourself in if you invest heavily in React Native. Sometimes I've heard of individual teams thinking that React Native is good for them, but you really need to take a holistic look at what its impact is, and what its continual maintenance is going to look like, because it's way, way more than a lot of people think it is.

[0:55:00.8] JM: Final question, we just did a couple shows about Flutter. Flutter is pretty cool. I don't know if you've had a chance to look at it. What do you think about the potential of other cross-platform frameworks, either for Airbnb or otherwise?

[0:55:17.3] GP: Yes. A lot of people have been asking this. Oddly enough, I feel like, Flutter with good reason has been very popular. The Xamarin folks are so passionate about their framework, so hats off to you guys for just loving it publicly so much. For whatever reason, that was notable for Xamarin.

For flutter and other frameworks, I think it's great that the community is trying to solve this problem, right? I think there's billions of dollars annually wasted writing the same thing on Android and iOS.

[0:55:49.2] JM: Completely wasted.

[0:55:50.4] GP: Yeah, exactly. Literally you're trying to write the same thing and if their difference is trying to understand where are they suddenly different. I think that this is a real problem and I think it's great that people are putting an effort to solve them. I think people really seem to be really enjoying Flutter. I think that it's really important to understand that even if they solve some of the technical problems, maybe the performance is better, maybe the IDE is better, the language you like more, you're still going to have a lot of the same organizational problems.

Plus, you lose the entire reason why at least we chose React, which is we have a lot of people who write React. At the time when we started React Native, we probably had three times as many people in the company and do React and JavaScript, than people who knew Android and iOS combined. I think that's a really important point. It being React is so beneficial, because when something doesn't work, you can Google it and you can find your Stack Overflow answer. You can find a Github open source project with thousands of stars that work. I think that these are incredibly important points that – and you would lose a lot of that by going to Flutter.

I'm not going to say – I wouldn't say that Google is definitely going to stop investing in Flutter, but I think that that there is more of a risk of Google stopping to invest in Flutter, because they build fewer internal apps with Flutter than Facebook has with React Native. I know Facebook has dozens if not, maybe even hundreds at this point of teams internally that use React Native.

[0:57:21.0] JM: Existential.

[0:57:22.5] GP: Yeah. Their app is huge. It may even still not be a huge percentage of their overall app, but it's enough that they have to continue to invest in it and the community has clearly invested a lot in it. That's really important. I also think that some of the Facebook published a really good blog post right around when we publish ours, about some of the improvements that they're making to React Native.

They're going to make it easier to write synchronous code between native and React Native. While that sounds it may only be useful in a niche case, it enable some extremely critical things, like for the first time, they can properly wrap, recycle view and UI collection view from React Native to Native. That is absolutely huge and it's going to solve one of the biggest pain points between the two.

Also, the fact that everything is wrappable, I I think that's a fairly big difference from Flutter. I built Lottie for Android, for example. Getting it to work in React Native is trivial. In an hour, you can write a wrapper and you can make it work. If you wanted to get that to work in something like Flutter with its own view system, then you would actually have to write your own renderer again. That could be a landmine for example, that could happen in the future.

It's really important to consider it holistically and what do you – it might solve some problems. It's going to introduce others and is certainly going to share a lot of the same problems as with React Native. On the Airbnb side, I think we really invested in React Native specifically, because of our infrastructure and our expertise with React. I think that that was our shot right now to do a cost platform.

I really don't anticipate us adopting Flutter, Xamarin or any of these cross-platform frameworks any time soon. Certainly not at least until React Native is actually out of the codebase, which is probably another year or two out.

[0:59:09.0] JM: Okay. Well Gabriel, thank you for taking the time to come on the show. This has been a really interesting and instructive topic. I think people really responded to your article and found a lot of value in it, and I think it's awesome that you were willing to talk about it. It's obviously like you said, divisive and a touchy subject. I mean, when it comes down to it, we're all just trying to build stuff and we're just trying to share information on how to build stuff, at least the engineers in the room. Maybe the vendors less so, the CEOs in some case less so.

From the engineers, we just want to know how to build stuff faster and more efficiently and I think your article went a long way to helping people better understand the pros and cons of using React Native. Thanks for writing the article and thanks for coming on the show to discuss it.

[0:59:59.6] GP: Thank you so much. It's always a pleasure.

[END OF INTERVIEW]

[1:00:04.5] JM: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting application performance monitoring and log management in one tightly integrated platform, so that you can get end-to-end visibility quickly. It integrates seamlessly with AWS, so you can start monitoring EC2, RDS, ECS and all your other AWS services in minutes.

Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that fuzzy comfortable t-shirt. That's softwareengineeringdaily.com/datadog.

[END]