EPISODE 1251

[INTRODUCTION]

[00:00:00] JM: When Uber was getting started, the company had to solve many cutting edge challenges for mobile. Uber's mobile platform needed to be safe and reliable despite the inherent unreliability of flaky mobile networks. In addition, Uber dealt with unprecedented scale in terms of marketplace dynamics and overall usage. In this episode, we talked with Gergely Orosz. Gergely is the author of *Building Mobile Apps at Scale*, a book he wrote. He previously worked as an engineering manager at Uber working on their payments experience platform as well as other mobile challenges. We discussed distributed systems, payments, and mobile technology, all the challenges that he overcame working at Uber. I hope you enjoy this episode.

[INTERVIEW]

[00:00:42] JM: Gergely, welcome to the show.

[00:00:44] GO: Great to be here.

[00:00:46] JM: You have a diverse software engineering experience. I'd like to start at Uber. You worked at Uber for four and a half years and you wrote a book about building mobile apps at scale. And I think of Uber as really an inflection point in how mobile software applications were developed. Because Uber faced so many unique challenges, there was just the volume, the marketplace dynamics, the safety dynamics of the application, the geospatial element. Tell me about some of the novel problems, the novel engineering challenges that Uber faced in a mobile context.

[00:01:33] GO: Yeah, so you do know a bunch about Uber. I mean, Uber started around right at the same time when modern smartphone app started. The iPhone was released in 2007. The first apps were in 2008. And I recall the first version of Uber was built by a team of contractors in 2010. And the first employee started to join, or at least the first mobile engineer joined in 2011. And then that's when it started. It was a first of its kind of this application. There were, of course, competitors. But one of the really interesting things that I found at Uber when I joined, obviously,

I'm not sure – Let me put it this way. I don't know any other company that has more mobile engineers than Uber has. When I joined, there were already maybe 300 iOS and Android engineers combined. And one of one of the things, the challenges came from this angle, it was tooling.

When I joined in 2016, we were rewriting the Uber app to using Swift. We rewrote it on Android, as well as on iOS, but the iOS one was a more tricky part. And we just kept running into limitations. We were the biggest app who chose to onboard onto Swift. And we had a lot of pain points. There was a Hacker News thread from someone who worked on a mobile platform team at the time. We ran into Swift limitations, binary size limitations. Build performance was a huge part. And related to this, Uber established the mobile platform team very, very early on. If I recall correctly, the fifth or sixth or seventh mobile hire already founded this platform team, and an Uber had a large in-house team building everything that we needed to get things done because no vendors were available. So this one from build systems, to experimentation, feature flags, analytics, crash reporting, everything, you name it, Uber built it themselves, because we have to. So this was I think one of the biggest challenges. As the mobile space was moving really rapidly, we have to build our own tooling to keep up to keep engineers productive.

[00:03:37] JM: So the big engineering challenge is more about the tooling that you had to build around shipping these mobile apps, or were there some like app specific mobile challenges?

[00:03:50] GO: I would say the biggest challenge by far – So the part that I saw the most, obviously, there's going to be some part of how you build maps. I was not in that team. I worked on payments and I built I worked on features. But I would say by far the biggest challenge Uber faced was a combination of tooling and build performance, and this had to do with how many engineers worked on the same code base. The biggest pain point was we had 50 to 100 engineers working on the same code base, building independent features, and we kept stepping on each other's toes. So a team who was building a feature in India would make a modification in this massive file that controlled the trip layout when you were on trip. They made a small change there. They shipped it. They tested it. It worked. And they broke another team's functionality in Mexico say.

So this was I think the biggest challenge, because yes, there were some additional things, battery consumption, networking performance, some of those things, and we had dedicated teams who looked into that, so like those were all challenges. But from my perspective, from someone who built features, it was just too many cooks in one application because we needed to build so many features. And that's actually one of the biggest innovations I think that Uber has brought actually to the broader community, which is when we wrote the Uber app, we built an architecture that actually is able to support hundreds of engineers working on the same code base called RIBs, which was a very opinionated architecture that defined how to build components, which then communicated in a way using immutable state, and kind of one directional data flows, that you couldn't mess up another team's components. And that architectural shift, we were the first app to do a massive rewrite after the company has had millions of users and billions of revenue, which is very unique. No other app has done this at this scale that I'm aware of. So Facebook, Twitter, they all kind of iterating on their save app. We had a fresh start about five or six years after the business was started, which was a huge deal. We spent a whole year writing that up.

[00:06:00] JM: Now, the timeframe that you were at Uber, was let's see. So you started in 2016, I guess? So I guess Uber was still dealing with some of the – With the intensity. Like I remember, I did some early – Or some of the earliest shows a Software Engineering Daily were about Uber. And like I remember talking to Matt Ranney, who I think was the CTO or chief architect or something of Uber. And my impression when I was talking to him was that there was just like a lot of sleepless nights. And that it was hard to do the kind of planning that was necessary and to do the kind of like long-term thinking that was necessary sometimes just because there were so many day-to-day outages and day-to-day problems. Was that your experience as well? Did you have kind of experience of the company just being kind of on fire all the time?

[00:06:56] GO: Yeah, so I was a bit more than four years at Uber, and it felt to me like four distinctly different companies. In 2016, for about a year, it was that hustling startup where everything was on fire. I had meetings at 1am in Amsterdam, got woken up at the middle of the night, our priorities changed last minute. We have to build this feature to get ahead of Lyft. The China business got shut down and suddenly we had to build some other things. It was go, go, go. And we had no clue. I had no clue what we'll be doing in a month, even a week, sometimes

even tomorrow. Let's just survive. There was that survival phase. And part of this was the app rewrite, which was a self-inflicted pain point. We had an internal deadline to rewrite the app and we push the company and everyone very hard. It was all hands on deck.

So my story was I was an iOS engineer before Uber. I did everything before, but I was doing iOS as my latest stint. I interviewed at Uber. I thought I was interviewed for iOS, but they put me on a backend loop, and I cleared it. So they offered me a backend job, and I was senior engineer. So it was whatever. Two weeks before I started, my manager asked me, “Can you do Android? Because we really, really need Android engineers.” And when I joined it was all hands on deck. Anyone who was living and breathing that was able to do iOS or Android have to change over and do iOS and Android to rewrite this app. So it was that that crazy phase. We also have to build tipping last minute in 2017. Lyft announced tipping. And until then TK said, “The two core things in Uber were we're never going to do cash or tipping.” My team in Amsterdam, we did cash. So that was gone. But all we were was built around the notion we will never have tipping. So we never added any support and the data schemas anywhere. And then TK turned on and said in two months. He actually told us a date when we need to launch tipping because he's going to announce it. And it was that firefighting. And things calm down afterwards. So like after 2017, the company shifted, and it's changed in many ways afterwards. But yes, for one year, it was that crazy go, go, go. Yeah, just madness. It felt like a proper startup that's on fire.

[00:09:10] JM: Tell me a little bit more about what your book tries to encompass, the themes of building mobile apps at scale.

[00:09:21] GO: So I started to write this book because Uber I'm pretty sure is or was the top three apps in the world in terms of complexity, may that be lines of code, features, number of countries supported, like actually supported having features deliberately for countries. Uber was always built around cities. So Uber, like it has distinct features sometimes just for cities. And even though Uber was mobile first, I always felt that a lot of the engineers who are not mobile engineers, and the majority of the engineers were not mobile engineers at Uber. My estimate would be maybe 10% of engineers were mobile engineers. A lot of backend engineers, web engineers, data scientists, etc.

I always felt that there was a bit of an assumption that mobile is simple, that the distributed systems, the backend systems are difficult. We have the data centers, the outages, and a lot of the leadership team had a backend background, and they understand exactly how those systems work, why they were difficult, why migrations take a lot of time. But mobile was a bit trivialized. I think, “Okay, we'll just ship it on mobile. It's just the frontend. How difficult can it be?” And people just somehow started to – And I kept reminding people all the time on like, “Well, here's how we do experimentation, and how mistakes are very, very costly on mobile. We had to ship everything behind feature flags. We have to be careful with rollouts. We need to monitor things like crashes, and a lot of other things in-app push notifications.” We had so many small things that all added up. And I found myself repeating again and again and sitting down with PMs and managers who did not do mobile before and explaining to them and they always had the, “Aha! I didn't realize mobile was this complicated.” And I was joking around with my product manager saying, “I should just write the 42 simple things that make mobile really difficult.” And she was saying, “Yeah, you absolutely should.”

So I started to drop the blog posts that I wanted to do first internally within Uber, but I never finished it. And I had a draft tucked away somewhere. And I decided to publish it after I left Uber to not leave it hanging. And I share the draft on Twitter about the table of contents. And I had a huge influx of interest from people working at Amazon, Twitter, Tesla, Grab, all of these companies saying, “Hey, we have the exact same issues. And can I review the draft because I want to see that?” So that's how the book was born.

I feel there's a divide between building simple and like normal apps, apps which are built by one or two engineers, they have your usual features. You have a lot of guidance there. So you just follow Apple and Android best practices, you use the libraries, you look at example of GitHub projects, and you're fine. But as soon as you go into these apps that are built by teams of more than 20 engineers, where you have to support more than 20 countries, you have more than a million daily users, every single one of these companies reinvents the wheel. Like I've talked with a lot of these companies from Robinhood to Lyft, and they tend to build some of the similar internal systems because there's nothing out there. And it feels there's a siloed nature in the industry right now. So hopefully, this book breaks it down a little bit. And as part of what I did, I collected the stuff that we did at Uber because I felt we were pretty much at the cutting edge of the industry. And I also reached out to people and did a lot of research on what are other

companies doing. And a lot of times we're doing similar things. And this is what this book is. So it's a collection of the practices that most teams use, and probably should use, who are building large apps. And the difficulty with large apps is you really want to move fast. You want to ship these features quickly. But now you do have all these large number of people to serve. So you need to be reliable. You cannot introduce regressions. And it becomes difficult. I mean, this is not too dissimilar to how you build distributed and backend systems at scale with large teams, except I feel that domain is a lot more explored. There's a lot more literature on it. For example, Google used to publish a lot of the ways on how they do it. They introduced the terms SRE and so on. And on mobile, we have a lot less of this.

And interesting enough, right as I published this book, the Mobile Native Foundation was founded by, I recall, Lyft, Uber and a few other companies who are creating discussion forums in the open exactly on this topic as well. So that's also an awesome resource to go to and to see how some of these other companies with larger teams are dealing and solving similar problems.

[00:14:06] JM: If you look at the backend world, you have community consolidation around Linux, obviously, and Kubernetes, and I guess you could even say AWS. In the world of mobile tooling, I don't get this – Correct me if I'm wrong, but I don't get the sense that there's the same element of community and like large scale evangelism and knowledge sharing. And it sounds like you're hinting at that. There's some siloing. If you go to Robinhood, and they're building the same thing, as if you go to Uber, they're building the same thing in terms of what needs to be done for building these large scale mobile apps. Am I understanding what you're saying correctly?

[00:14:53] GO: Absolutely. And as part of writing this book, I did a lot of research. So I've talked with a lot of people. A lot of people have reached out to me. It's 100% of the case. The really interesting thing about this, I feel it goes back to the premise of my book. Most companies do not understand the complexity of mobile. A lot of companies are starting to get it that it's important. I've talked with many companies, I cannot say the names, but what a lot of companies are seeing is their revenue stream is shifting towards mobile. So I'm not a large bank. I can't tell their name exactly. But they're seeing that the majority of their customers are now using iOS and Android. And the customers who do have the smartphone app installed are generating more profit. They have a lot less churn. So what this bank is doing, they're investing

a lot more, they're retraining some of their web engineers to become mobile developers. And this is just not even a market leading bank, but they have a mobile team of 130 native engineers, and they need more. But they're still this notion from leadership, and anyone who has not worked on these mobile apps, even web engineers that, well, mobile should be pretty simple.

And interesting enough, there are some vendors who are building specialized tools, there are CI vendors, there're crash reporting vendors, and some of these are venture funded. In fact, multiple of these are venture funded. But even the VC firms are very skeptical of the size of the mobile market. I feel everyone is under estimating it, and we are in the middle of the shift. People are starting to use mobile as their primary use for anything. There are those apps there's. It's hard to break in, but those absolute will keep living. And for the day-to-day convenience things and things that move money, mobile will become a lot more important.

So I feel that I am very surprised that the community is not as strong, that the investment is not as big, that decision makers, I'm talking about CTOs, just continuously underestimate on both the complexity of mobile and also the impact of have a strong mobile engineering team. The companies that invest, I'll tell you, some of the top apps in the app store, Uber, Reddit, Pinterest, etc, they have dedicated teams of three to four engineers just working on app performance. And you might think, “Well, that's silly.” No, it's not. They looked at the numbers and they know that the if they speed up, if they make their Android and their iOS apps load within half a second or one second for low-end devices, or reduce latency and startup time and networking by milliseconds, they're generating a lot of revenue. It pays for that team 10 times over. Most of the industry has no clue about this happening. So there is an invisible divide happening. Hopefully, this book will actually shed light on this a little bit, but it's not going to fix it.

[00:17:45] JM: Can you talk a little bit more about the internal tooling that gets built around these mobile apps at scale? Like, one thing that comes to mind is since you're in Uber, well, I guess this is a Lyft tool. But I remember talking to Matt Klein about Envoy Mobile, which was like this this whole agent for doing networking, like basically mobile networking. They built this really richly featured networking tool called Envoy Mobile. And if people are curious about that, they can listen back to that episode. But I just love to get more of a vision into that mobile tooling that was built at Uber.

[00:18:31] GO: So for better or worse, I guess worse for the community, but a lot of the tooling that Uber built is not open source, mobiles I'm talking about. The one thing that is open source is the RIBs architecture. And there are some other tools like Nanoscope, which is a performance tool. There's also NullAway, which is a null checking tool. But at Uber, the whole stack, everything is internal tooling. So as a mobile engineer, when you join the company, the first thing you do is you're going to check out the repository, make a change, make a simple text change or something, push that diff, because Uber uses fabricator, and then you'll want to merge that into main. And you're going to see the Submit Queue. So Uber built this internal tool, which is the agent that takes care of running all the tests, UI tests, unit tests, linting, etc., and merging it into master. And that might sound simple, but it's not because the problem that Uber faced very, very early, is every build takes about 30 minutes to complete mobile build end-to-end, running all the UI tests because most of the UI tests and snapshot tests take a bunch of time. But Uber was already at the scale where every hour we had maybe 10 or 20 or 30 merge requests, which meant that these needed to be run parallel. But if you run them parallel, a lot of times they will cause conflict. So you merge one then the other one was thrown back, which effectively meant that – I was there in 2016. You have to wait three or four hours and restart your builds three or four times just to be able to commit this.

And so a team, the developer platform team built this tool called Submit Queue, which used this clever way of trying to predict conflicts between the changes going in. And they wrote a really cool paper about it. They used probability graphs. I don't know the exact term, but the paper has this. And they build the Submit Queue, which was able to merge 90 something percent of the changes in one go. It just kind of figured it out. And this was essential for Uber to be able to scale builds, meaning support hundreds of people working on the same code base. Uber also moved to a mono repo really early. iOS and Android have their own mono repo. So all the mobile code is checked in there and you build the separate apps from there. And at one point, Uber had a lot of apps, 10, or 12 different apps when there was still Jump, and Uber Rush, and a lot of other lines of businesses as well.

Also, there's a custom crash reporting tool, there's a custom sanity testing tool, there's a custom localization tool, there's a custom – The experimentation and feature flag tool is also custom. Now, one of the reasons a lot of this is not open source, because it's tightly integrated with each

other. So when you're working with a localization tool, if you make a localization change, it will typically propagate across all of the properties, so the web, the backend, and so on. When you do an experiment, or you change an experiment, you can see all the experiments that are working at the same time and the data science team can mine that data from anywhere within the company. So a lot of it is tightly coupled. And I think it goes back to the fact that Uber was and is mobile first. And they just built, or I guess we built while I was there, all this custom tooling that gives it an incredible edge in being able to run a very localized and detailed experiments and have control over the full stack.

Still, I think, for example, the crash reporting functionality at Uber is more advanced than almost any vendor could offer. And that's one of the reasons they're not using vendors over time, that edge will disappear. And I'm sure we were will start to use some third-party solutions, but it's still there. And a very interesting thing on Uber on why – I always asked us, “Did we need to build this?” But Uber is built around cities, and all the tooling is built around cities. You can narrow down to city level. There is no vendor in the world which supports city level targeting. And even if they do, they don't have the sophistication that Uber has to actually map exactly based on IP or other device data. So that's another advantage that Uber I think does have compared to their competition. I'm not sure exactly how Lyft is doing it. I'm pretty sure they have a really good engineering, but some of the other competitors, may that be Bolt or Grab, they either need to build this themselves or try to use some vendor and kind of hack away with it.

[00:23:00] JM: How did you manage the understanding of all the different tools that were available in the company and the relationships with those different tooling teams? Like you're an engineering manager for much of your time there, and I imagine there's a lot of communications overhead that goes into understanding what is available in the company. Especially if you're talking about there's like 10 different apps within the company and it's probably maybe the Jump app has crash reporting already implemented in it and you don't know whether or not there's a team that you can go to and ask for crash reporting. I just love to know that the dynamics of – Because I worked at Amazon for a while, and I just remember, it's so hard to know even what is available within a large company like that. So how does the tooling get indexed? How do you find out what is available?

[00:24:05] GO: I mean, obviously, a lot of it was tribal knowledge to start with. Some of the tools did link to a lot of the other tools. So I'll give you an example. The kind of heart and soul of Uber was what we called Metro, which is the visualization of this Submit Queue. So you can imagine, if you typed in metro.uber.com, or the internal address, you were directed to a UI where you saw a build train. So it had like city iOS rider app selected. And you saw that right now the latest version has rolled out to 50% of people in the marketplace. And you could switch a toggle to look at the different versions where those are, and then you can switch to different apps on where those are. And this app had kind of gates. So there was a square that showed the app store rollout, and before that it was the internal roll out. And there was a bunch of arrows which all were a step that needed to be green on the order to proceed.

So for example, in order to roll out an app, you needed to have the localization done, all the UI testing to be done, no manual regressions. All hot fixes need needed to be there, and so on. And all these arrows were a link to one of these tools. So you could take a look at the status of all the hot fixes. You could take a look at all the crashes just by clicking through. So that was pretty clever. So it was somewhat in one place. And the other part was that Uber had one mobile platform team. It was a really big mobile platform team, but there was only one of them. So when engineers on boarded, that platform team maintained the wiki, and it did have a list of these internal tools.

So I felt it wasn't that bad. It did take a while to onboard at Uber. So I had engineers on my team who joined. They were amazing iOS and Android engineers outside of Uber. A lot of them Google developer experts, etc. And after a year one of them came up to me and said, “Hey, Gergely, let me tell you, I don't feel like I'm not an Android engineer. I feel I'm a RIBs engineer, and a hotcakes engineer, and like an Uber internal tools engineer.” Because Uber, the platform team, I don't even know the details of how we did networking, because there was a networking team in the mobile platform team who maybe they did something similar to Lyft’s Envoy Mobile. They took care of the networking layer, and they optimize it, they measured it, they sample performance. It just worked. You just use that component. And we had a lot of these low level things that you just used in the app, and the platform team made sure to optimize the hell out of it across all the different devices, OSS, etc.

So in that sense, it worked pretty well. I feel it was a lot more chaotic before 2016, before the app rewrite. The app rewrite, I cannot stress enough how big of a cultural change rewriting the app was and introducing this concept of RIBs, a router interactive builder, this concept of components. It was a very opinionated way to build software and it deliberately separated concerns. It separated teams very efficiently. If you wanted to change something that was not within your team's component, you have to go through a lot of steps, including sometimes mobile platform reviews. On one end, it frustrated engineers, because you suddenly lost a lot of the freedom. You couldn't just change kind of core parts of the app or anything. On the other end, it reduced bugs. It improved stability, and it kind of confined teams into their own little space. And so the contracts were super clear. And this is the thing that I think a lot of companies should take from Uber. And Uber had the opportunity to rewrite their app. And we made it very opinionated. We made the contract super clear. I think that helped Uber sustained growth and being able to ship features without breaking things randomly and without having to rely on an insane amount of manual or automated tests.

[00:27:59] JM: So you did have an insane amount of manual and automated tests.

[00:28:03] GO: I don't think we had an insane amount. I think we had a good amount. Definitely, now that I'm talking with other teams, it seems some companies have I think more tests than Uber. I don't know the exact number that we have. We were big on unit testing. Everything was unit tested everywhere, everything that was business logic. I always had screenshots test. Android did not have any screenshot tests, which was interesting. It's a very divisive topic. We have very few end-to-end and UI tests. And we did have manual and exploratory tests that were run either by the teams initially and then later by dedicated people who are doing these things.

What Uber was really good at though was monitoring and alerting. So we had a big beta program. Anytime we saw aggression that was caught pretty early. And we had stage rollouts. And so the experimentation platform was, I think, first class on mobile as well. So we caught most of the issues during either testing or rollout. So it didn't get to customers, or at least not many customers.

[00:29:06] JM: Now, I did a bunch of shows on Facebook. And one of the big frustrations at Facebook was that release engineering for mobile apps was terribly difficult. Now, I think things

have changed since – I was interviewing somebody about that. Basically, how Uber did – Or how mobile release engineering worked maybe like seven or 10 years ago. I imagined things have improved somewhat. Can you tell me the state of mobile release engineering? Are there good CI/CD patterns? Or like what do you do for high quality release engineering?

[00:29:44] GO: I mean, I think it's improved to how much you can improve what the app store limitations being in place. It's still painful, and the release engineer's job is still terrible, I think. So I feel at this point if same old same old. So from a developer's perspective view, you make your change. And every Monday or every certain day, I think they keep changing it. But at some point it was every Monday afternoon, there's a build cut. And you know that if you make that build cut, then you're you'll be in the build candidate, which will go out in about two weeks for. For a week, this build cut bakes internally, so it goes into internal beta program. Employees had a large amount of Uber credits. I think every employee would get credit for 15 times the average order in your city. So you can order 15 eats or rides pretty much. But in return, you have to use the beta app. And so this would mean like that there will always be 1000 people testing it.

After a week, if there was no regressions, this will go to the app store. Now, there was always a regression. So you typically – If you're one of the teams who still got a regression somehow, you've got a ticket assigned to you, which you have to answer, "Does this need to be fixed right now, because we caught this issue?" It might be manual testing, or localization, or something else. And if so, you needed to do a beta fix, which meant that you had more than 72 hours until the final build cut. Or if it was less than, I think, 72 hours on an alpha fix. And these had – As a manager, I have to approve these fixes. So we always decided, "Is it worth doing a fix? Or do we just turn off this feature and not roll it out?"

And then once this cut was done, then the rollout started in the app store. In the Google Play Store, there will be a stage rollout. In the app store, you can't really do that. So it was just replaced. And we saw the uptake of the latest app. The frustration with this that will not go away until we can release them instantaneously or Apple changes its policies, that if you made a change, it took about three weeks, two to three weeks to go into production, which meant that you knew that you couldn't just fix things magically on the app, which meant that every single change needed to be behind the feature flag, which also meant that the app was polluted with feature flags that were sometimes just left in there.

Now, one thing that every single company, including Uber, realizes, well, that's not great, because we just ship instantaneously like in the web. And following Apple and Android guidelines, you cannot ship business logic to the app. But you could embed web pages. Or you could do a backend-driven parts of the application where you send some sort of JSON or XML or metadata. And Uber did this as well. We built an internal tool, which doesn't come with a whole set of different drawbacks, but I feel the industry is working around that. At least these apps are trying to work around the fact that Apple is gatekeeping, that they want to check every single thing that goes into the app. And it's kind of artificially limiting your ability to ship functionality or sometimes even fix things.

But I think it's been accepted. We know how it is. And I felt that Uber after a few years, it's just everyone knows what to expect. Everyone know how things will be going out. And this is also why if you if you follow Jane, I think her name is Jane Wong. She's a, I guess, researcher on her own time, she has a hobby of opening up the apps are in the app store and looking for the feature flags, turning them on, and tweeting out features that are going to be shipped at every single app, because the whole industry works like this. We ship things behind feature flags. We don't turn them on until we're very certain that they work. So actually a lot of the feature functionality for Uber, for Facebook, for Twitter, it's in the app, except it's a bit tricky to look at it. But some people look at it already.

[00:33:37] JM: Do you have any general advice for people out there who are managing mobile teams? So just, obviously, there's general management advice, but I wonder if you have domain-specific mobile engineering management advice.

[00:33:58] GO: What I have benefited a lot from is two things. One is get your hands dirty. So if you're managing a mobile team, understand exactly how the chain works. I benefited greatly from – I was a mobile engineer for six months. I was an Android/ iOS engineer. On paper, I was an Android engineer, but also I was familiar with iOS. And so I pushed code for six months. I wrote the RFCs. I made the code changes, reviewed the code. I fought with the tooling. I made hot fixes. I did all these things. I saw how long my changes to go out. I monitored the feature. I think as a mobile manager, you need to get your hands dirty on how these things work. Either just ship a feature into production, or you can just observe, but you should know exactly what

your team is working on. What tools they use. What the frustrations are. So have an ear on the ground. That's the first advice. The second advice is don't just look at mobile as a silo. I feel the mistake that a lot of people do, especially if they're longtime mobile engineers, a lot of people have been developing iOS or Android apps since 10 years now and they're on paper, senior engineers, but they never looked at the backend or the web. Because I feel mobile is not done in isolation. There are very real tradeoffs. And you should ask yourself, when you're building a mobile functionality of, “Well, should we experiment on this with on a mobile, or should we maybe ship it on the mobile web where we might have fewer customers, but we might be able to ship things faster?”

At Uber, I created the first – I was on the payments engineering group. My team built pretty much everything that is in the mobile app that has to do with paying things in the rider app later, in the driver app and in some part of the eats app. And I found the same web team where we started to do web payments, we consolidated before that there was no team who was doing payments on the web, and we did that. And it was really, I think, enlightening to see how the web was so much faster in so many ways. Obviously, a lot more of our customers were on a mobile. But this was very interesting. And all the teams who reported to me iOS, Android, and web engineers, they were on the same page. So I guess maybe my third advice is don't accept having silos between Android, iOS, and even potentially the web. These teams, customers don't care. They expect the same thing to work in similar ways. The back end team ideally shouldn't really care if it's Android, or iOS, or web. They should work on API's that hopefully work for most of them, although mobile has different things. So if you have silos between iOS and Android, start to break them down in different ways.

One thing that we were always really good about and our architecture RIBs really helped, is iOS and Android always work together. So on every feature, we had an iOS and an Android engineer, or two of them, and they just collaborated. They shift at the same time. They reviewed each other's code, because it turns out reviewing Swift code or Java code, and later Kotlin code is not that big of a deal. And sometimes they also wrote. They sometimes they switch platforms and they just wrote code on the other platform. So I would say don't put yourself in a silo.

[00:37:15] JM: What was the biggest engineering problem you remember having to solve while you were at Uber?

[00:37:21] GO: The biggest engineering problem. So there was – Okay, I'll tell you the biggest one. This was when we started to – When I took over the rider payments team. So our team was building everything in the rider app that was payments. There was about 13 or 14 different ways to pay from credit cards, to PayPal, Apple Pay, Google Pay. Those are the ones everyone knows, to Paytm, GeoPay, Alipay. These are basically regional payments that were either in one country or in one region. And we built this in a way that we tried to build it in a bit of a reusable way so that later someone else could use it. But our only customer was the rider app. And sure enough, as Eats started to grow, Uber Eats was another startup within a startup. I like to think of Uber as this common collection of startups. They wanted to move fast, and they came to us saying, “Hey, can we use your credit card implementation?” And we're like, “Oh, sure. Here's this library. We just need to make this small changes to it. It'll probably take a month. They're like, “Ah, okay.” And they never came back. They built their own implementation, because they were like a month is too long. They needed to move fast. And fast-forward 12 months later, Uber Eats had their own implementation of all the different payment methods that we had. And it was taking them a lot of maintenance just to keep up. We had some compliance changes and decided that whenever you pay with a credit card, you have to have a special checkbox for this or that. And now the Eats team has to build that instead of shipping their features. So the requests that we started to get is could we just not have one library? Like could we have a Stripe-like library that you integrate, but with an Uber? Because we didn't use Stripe for like various reasons, for example, because we're using competing payment methods with Stripe. But the idea was could we have a library? A payments library that could just drop in and SDK at Uber, internal like stripe SDK within an Uber? Like a payments SDK? And this turned out to be such a big challenge. Because there were so many teams using it, everything was changing all the time, everyone wanted to get different things done. Each team just wanted to move fast. We have tons of hacks inside. And it turns out that when you build a library that you think is going to be reasonable, but you don't know who's going to use it. It's not reusable.

So this, what called payments framework that we built for the rider app, it was completely not usable for – Or it was very difficult to use for Eats or for the driver team or later for the line team, etc. So it was a humbling moment, because like I used to be very much of this couch architect. When you read about a company doing something seemingly silly, I don't know, Microsoft delaying Windows Vista by a year and it's still a terrible product, right? You're just there saying,

“Oh, they just didn't know what they were doing.” And here I was, where if anyone else from the outside said, “Well, Uber can't even create like one payments library. Like how incompetent the people must be.” That's what I would have said. But it was not true. We were very competent. Except when you're in a large organization, you have a lot of different teams who are shipping all sorts of different things. It's just hard to build something that keeps changing. And we pulled it off in the end. It actually took a few engineers to step up and just make some opinionated decisions to go in and integrate this library.

And the biggest problem, the reason we couldn't really pull this up quickly, is a lot of money was dependent on this. Specifically $65 billion dependent on this thing working at all the time. We couldn't break it. And that was the biggest barrier. It was payments, right? You cannot break payments. And if you do, that's very expensive. I've seen outages where it cost the company millions of dollars, and you just don't want to do that. So this makes it even more difficult. So yeah, I was just very humbling to see how difficult some seemingly simple things. You just say, “Payments SDK, how difficult could it be?” And it was, it was difficult. And by the way, the engineering part was not that difficult. It was more of the organization, the API's, the migrations, and doing it without impacting any customer. And the customers didn't see anything of this.

[00:41:32] JM: That's really interesting story. As we begin to wind down, I'd love to know any like larger scale reflections on your time at Uber and how it compared to the other companies you've worked at. You spent a lot of time at Skype, Skyscanner, and you've worked at JPMorgan. I just love to know your reflections on Uber relative to the rest of the industry.

[00:42:00] GO: For me, Uber was a really awesome experience. It was by far the most fun that I've had and the most challenges and the most professional growth. I spent four years there. I feel as if it would have been 10 in terms of learning-wise. And I've had many learnings, but some of the key things that stand out is one is the concept of ownership. When I joined Uber, Uber had this cultural value. They have 14, but one really stood out, which was be an owner, not a renter. And as soon as I entered the company, this was established already. People around me were celebrating. Someone did something – It was basically if you see a problem, don't complain, fix it. And people did it all the time. And they were being celebrated for it. So like teammates will say like, “Oh, great job being an owner, not a renter.” And it was magical,

because Uber was held together by duct tape. Seriously, I was shocked at how bad things were. But instead of people complaining, they kept fixing it. And it was very empowering.

I saw very junior engineers join out of college, or like one or two years of experience. And two years of doing being an owner, not a renter, keeping fixing things, not accepting that things are broken, going over to the team committing into a code base, they become really experienced engineers with great judgment. And this is an environment which I would want to recreate wherever I go. And I think it made a huge difference. It's very different. Like I don't think you can – It's hard to transform a company that doesn't have this in their DNA. But Uber did have this.

The other part was the transparency. And some of the interesting processes that Uber put in place really early. Uber didn't have many processes. But one of the main processes that was in place from the early days was RFCs, requests for comments. Whenever you wanted to build a new feature service, or modify something that was not trivial, it took more than maybe a month of work, you wrote this document, and you sent it out to pretty much all of engineering. There was an email list where everyone could subscribe and everyone was subscribed. And I would have thought this would never work. It worked beautifully. It worked really well until maybe two or 3000 engineers. People chimed in, who were interested in this, it created transparency, and it also forced people to do some planning upfront, which I think engineers are really bad at doing. You just have things in your head and you start coding and then you go back and erase two weeks of work. So this was an interesting one.

And the final part was I talked with the first five engineers at Uber. I interviewed them because I wanted to write an article. I never wrote that article. But it was incredible to me how those first few hires shaped the culture. So it turns out that this RFC process was introduced by, I think, employee number three or four who was really a junior guy with like one or two years of experience, but he read all these books about architecture. And he had this idea that we should just write things down and he just convinced the team when there were three or four people. And from there on, it was a thing.

One of the early – The first mobile engineering manager was actually an Apple employee who saw the Uber app. And he liked it. He started to use it. But he was pissed off on how the cars would not go on the street. They would just go over the – It was going in a line. And he came

over, because I think he was working next door, he came over and he talked with TK saying, “Hey, I want you to fix this.” So they had some really good leadership join early on as well. And those, like, I think a lot of the Uber culture might have been shaped by some of these people just kind of doing things early on, and it just stuck, which is very, very interesting. I almost feel it was a bit random that some of these things happen the way they happened. So yeah, those are the key learning.

Ownership is such – If a company can keep on this feeling that people feel that they're owners and they can change things, it goes such a long way. And yeah, this planning documents, they do work, especially when you start early on and it becomes the DNA of the company.

[00:46:05] JM: Awesome. Well, Gergely, anything else you want to add before we close off?

[00:46:11] GO: Just one small plug for this book that I wrote. It's actually a completely free book for until the 31st of May. So it's the full book with everything in it. And it's because I was able to do this deal where I secured some sponsors to get reach. So if you're interested in mobile apps in any point and you want to get a read of it, grab it while it's free. Later, you can also get it on Amazon or other places. And yeah, if you have any feedback on mobile access skill, I'd love to hear from people. You can find my email and contacts on my web page. And I also write a blog called The Pragmatic Engineer, where I share some of my ideas about engineering, about RFCs. I wrote an article on that. So just check it out if you're interested in reading some of these thoughts.

[00:46:57] JM: Awesome. Well, thanks again for coming on the show.

[00:47:00] GO: Thanks for having me. It was great.

[END]