EPISODE 314

[INTRODUCTION]

[0:00:00.6] JM: Parse was a backend as a service company built in 2011 before being acquired by Facebook in 2013. Building a backend as a service for developers requires walking a thin line between giving engineers lots of control and preventing those engineers from shooting themselves in the foot.

While she was at Parse, Charity Majors learned about the operational burdens of managing a service with really high uptime requirements and deeply technical edge cases that could take down a user's entire system. Charity took the lessons in systems engineering that she learned at Parse and cofounded Honeycomb.io, a service for observability and monitoring. Honeycomb is described as a tool for your systems like an IDE is to your code, and we get into that and what it means in this episode.

Parse was eventually shutdown because the service did not have a place in the strategic plans of Facebook. Charity and I also discussed the lessons that she learned from how the Parse acquisition panned out. This was a useful conversation for anyone who is considering selling their company or acquiring a startup, because this was a case in which the acquisition didn't exactly work out in the best way that it could have. It was nonetheless a super interesting episode filled with lots of insights. For that, I'm grateful.

I think you're really going to enjoy this episode with Charity Majors.

[SPONSOR MESSAGE]

[0:01:42.6] JM: You are building a data-intensive application. Maybe it involves data visualization, a recommendation engine, or multiple data sources. These applications often require data warehousing, glue code, lots of iteration, and lots of frustration. The Exaptive Studio is a rapid application development studio optimized for data projects. It minimizes the code required to build data-rich web applications and maximizes your time spent on your expertise. Go to exaptive.com/sedaily to get a free account today.

The Exaptive Studio provides a visual environment for using back end, algorithmic, and frontend component. Use the open source technologies you already use, but without having to modify the code, unless you want to, of course. Access a k-means clustering algorithm without knowing R, or use complex visualizations even if you don't know D3. Spend your energy on the part that you know well and less time on the other stuff. Build faster and create better. Go to exaptive.com/sedaily for a free account.

Thanks to Exaptive for being a new sponsor of Software Engineering Daily. It's a pleasure to have you onboard as a new sponsor.

[INTERVIEW]

[0:03:11.6] JM: Charity Majors is a cofounder of Honeycomb.io. Charity, welcome to Software Engineering Daily.

[0:03:17.7] CM: Thank you. Happy to be here.

[0:03:20.3] JM: It's great to have you. Observability is this word that seems to have become a buzzword of the day. In our industry, I think that the buzzwords are actually quite important, because under the surface, whenever a new buzzword comes out, it usually means that something new is actually going on. We see this with microservices. Microservices is a new word used to describe the same old service-oriented architecture, but the word describes newer ways that we are implementing services or at least signals that there are some new ways.

I think the same is true for this "observability" term. It basically means "monitoring", but there are some newer implications. What are the aspects of our modern systems that have changed the best strategies around monitoring and why have we started using this word observability?

[0:04:11.8] CM: I think that what we're seeing now is a giant trend towards software engineers needing to own and operate and be involved in the long-term, from architecture all the way out towards maintaining and even asked to be on call for their own systems.

Another buzzword term that has irritated me and many others for a long time is “full-stack”. When in an interview and somebody tells you they’re a full-stack developer, I often will just roll my eyes and snap, “Okay. So talk to me about the chip you designed while you were also generating the UI and UX.” It’s a meaningless phrase, but it also points to something that’s real, which is that, increasingly, either you go way deep on one thing or you have to go broad.

Ops isn’t going away, DBAs aren’t going away, but we’re increasingly on the other side of an API, and you can’t just outsource all the understanding, and wisdom, and intelligence to a human who’s sitting next to you, or a human that you can even get a hold of. Observability ties into this in a big way, because from the beginning, it isn’t the case that you can build a system, look at it, predict how it’s going to fail, and then someone in ops will write monitoring checks for you.

Observability is instrumentation from day one. Every poll request, you check the syntax, you check the — All of these things that were used to checking, you need to start checking each other for instrumentation. How observable is this? How much you’re going to be able to predict what happens and how to model it mentality and how to fix it from the outputs that you’re getting from it when certain state changes occur.

[0:05:55.3] JM: There are some fundamental technical trends that are leading this imperative about different observability change. Whether we’re talking about containers, or cloud architecture, or wide use of third party APIs that we don’t know how they’re instrumented. What are those salient changes and how are they driving the need for new tools?

[0:06:22.7] CM: There’re a couple of things. One is things are getting better. It’s kind of like this is the story of capitalism, right? The person who’s best at baking becomes better and better at baking and pretty soon they don’t so anything but baking. We’re seeing that happen; people who are good at databases, do databases and soon they can be better DBAs that you can and you give them money for that.

So that’s, I think, reaching a certain convergence with this whole cloudification; push everything to the cloud, everything is a SaaS. There are so many people in the world who can do everything better than you or I can. I went to my last datacenter to flip a hardware switch at 3:00

AM sometime in 2008 or 2009. I never plan on doing that again. I use an API. There are still hardware engineers in the world, but fortunately I'm not one of them and I don't have to pretend to be a really crappy one.

This is happening out in much more intangible higher level services; Qs. People are just plugging together all these components to realize their vision, and their vision is not, "I'm going to master RabbitMQ, and MongoDB, and blah-blah-blah." Their vision is, "I have this app, or I have thing, I have this thing that I want to make real in the world and all I need to understand about these components is how to glue them together at a high level and how to think rigorously down from the product."

We saw this so much at Parse. I feel like working at Parse, I was so fortunate to have kind of a front seat to a lot of things that were happening in the industry from the perspective of these either Mom & Pop shops wanting to build an app for their own sake. Maybe they build a team afterwards, but they had to start from this perspective of, "I understand what I need. I understand what my customer needs. I'm going to build that. Everything that is not core to that, I offload."

The other scenario is smaller teams, or smaller experimental, maybe, visions within a large organization, it enterprises like try an internal iOS app, or maybe they're spinning up container, 'cause they think this might be the future or it might be a vision of the future that applies to them, but they're looking on this 5 to 10 year horizon, so they're trying it. Not everything that they try is going to become core to their model, but there's that baking-in process. I think that both of those are becoming a really common trend.

[0:08:47.8] JM: You worked at Parse, which was a backend as a service before it was acquired by Facebook. Describe what it was like to build a backend as a service in — What was it? 2009? 2010?

[0:09:01.0] CM: Yes. It was practically hard, because think about any workload that you might need to support as a backend engineer and then multiply that by N, where N is as many customers as you have signed up and you're trying to make that a very large number. So you

basically have to treat them all as — You cannot have to care about anyone of them in order to make all of them succeed.

Not that means you're going to supporting write heavy apps, read heavy apps, social apps, geo apps, any type of apps, apps that need 10x overnight , because they got featured on the iPhone, or the iTunes store, and then 10x again, because they did marketing, and then 10x again, and then Dropbox from that. It has spoiled me forever having to care about one company again in my life. It's so hard, and so chaotic, and so interesting, and so fun.

Going from building — Running databases for — We had over a million apps when I left, to running databases for a single company just seems like — It seems like a solved problem, and I know it's completely fair, but it's really hard to go back to wanting to care about one company.

[0:10:09.9] JM: What were the observability challenges in building a backend as a service?

[0:10:14.2] CM: Oh man! I love that question. It's so simple and yet it describes four years of my life. Take whatever observability problems you have as a single company, "What is my latency like?" "What is my throughput like?" "How are things changing?" "What features are people responding to?" then multiply that.

Here's an example of — I will say that around the time that we got acquired by Facebook, I would say that we had built what was effectively an undebuggable system. We were using best of class tooling in terms of open source. We have a couple of engineers who care passionately about that sort of thing. We were using auto scaling groups that would scale up, scale it down, and it would like immediately populate all the graph, all the dashboards, all of the service level views. Yet, it was all but impossible to track down simple questions, like a user would write and it'd be like, "Parse is down." I'd be like, "Parse is most definitely not down."

I have this wall full of dashboards that shows me exactly how not down Parse is, but they don't care about that. I'm actually losing credibility every moment that I tell them that what they're seeing or experiencing is not true. Maybe — To be perfectly honest, almost half the time, what they're seeing isn't true. Maybe they forgot to turn on their WiFi. Maybe they made up — They

updated their SDK and thy're black holing some error that is telling them that they're not including the right key.

If I'm just telling them, "You're wrong," that's not good enough. If I tell them, "Your requests are not hitting our edge," that's more credible, but harder. That's the easiest. The easiest category of problem is saying, "I don't see your requests."

If they are hitting us and they think that we're down, then why? Examples of answers might be they have some subtle bug either on our side, or in the SDK, or in what they're sending. Maybe they're sending an empty blog where they thought that they were sending a full blog. They'll just blame that on Parse, especially when we were doing our rewrite from Ruby to Go. Maybe they had hardcoded some expectation, like the type of this thing that return is supposed to be null.

In Ruby, they just assume a lot of things. In Go, you make everything explicit. Maybe we're returning a zero and that crashes their app. Maybe their crew was doing full table scan, something that they were doing when a user logged in and they went from a thousand objects in the collection to a thousand and one and suddenly everything is timing out. All of these things, they're going to be, "Parse is down." The big challenge for us was to make every single one of those an infinite array of possibilities, something that we could answer while conversing with them.

[0:13:08.3] JM: The challenge to me seems like that you have to build a backend as a service that is flexible enough to be powerful for a variety of highly technical users, but also prevent these users from shooting themselves in the foot, and this is a difficult line to balance.

[0:13:28.7] CM: It so is. We started out — I don't think that we ever got less — We never got more flexible in what we're willing to accept. This makes sense. When you're starting out as a platform and your only goal in life is to please the users. You're going to offer them things that they ask for, like regular expressions in your database — Colossal mistake.

We spent years trying to roll that back. We removed it from the docs. We stopped publicizing it. We started telling people not to do it. We severely weight limited it. Especially, mobile apps, you can't just stop supporting it, becase they might not have published an update to the app store in

one or two years and still be successful. There are so many baked-in assumptions that you can never stop supporting. We would like softly deprecate it.

Yeah, reason this is hard is because everyone of your users will have a different calculus for what they care about and what they're willing to accept. They want all these features, but they don't see the cost of them in a backend side 'cause they're not exposed to the pain. They just want to them, 'cause everything feels costless.

Parse went out there and was pretty aggressive about, "You don't have to know anything about your backend." Which was great from a marketing standpoint and just fundamentally untrue from a technical standpoint. I could always tell the difference in software engineers who had any concept of how backends worked or not just based on what they would write.

[0:14:50.2] JM: My sense is that — I get this from watching some of your talks, so maybe this was a more extreme version of your views, but my sense is that you have some skepticism of platform as a service, or at least how platform is a service is viewed by some developers, or how it's sold by some vendors. Where does your skepticism with platform as a service lie and — Yeah, I don't know. Help me understand the measure of your skepticism.

[0:15:19.5] CM: I get that. I'm mentally rewinding to some of my talks you may have seen where I have —

[0:15:24.9] JM: The Serverlesconf.

[0:15:27.1] CM: That was a special creature. I am deeply in love with platforms and services. Honestly, I love the category of problems. I love the engineers who are blazing these trails, I do. I have at times felt some frustration with the mismatch and how they're sold and the realities of life. The fact is that a speed of light is what it is. We're not going to change the fact that a full table scan is bad and doing 10 of them is worse.

Surfacing that information to engineers so that they can make good choices is really hard. You're assuming a level of technical literacy that may or may not exist. You are pitching it to a

level that they may or may not be at. The more knobs you're giving them, the more it's not as easy as it might seem.

For example, think about the AWS Mobile SDK. It was always "better" than Parse was. It was all way more fully featured. It was more powerful. It gave you more, as you say, rope to hang yourself with. Nobody loved it, because it wasn't opinionated. It didn't have this opinionated view of the world that, "This is how it works. This is how you can use your intuition."

At Parse, the steps that we aggressively took was you don't have to know anything. I definitely had my frustrations with that even while — I have to admit that that's why people found it lovely and delightful to use.

[SPONSOR BREAK]

[0:17:04.0] JM: Are you ready to build a stunning new website? With Wix.com, you can easily create a professional online presence for you and your clients. It's easy. Choose from hundreds of beautiful designer-made templates. Use the drag and drop editor to customize anything and everything. Add your text, images, videos, and more.

Wix makes it easy to get your stunning website looking exactly the way that you want. Plus, your site is mobile optimized so you'll amazing on any device. Whatever you need a website for, Wix has you covered. The possibilities are endless, so showcase your talents. Start that dev blog detailing your latest projects. Grow your business and network with Wix apps that are designed to work seamlessly with your site, or simply explore and share new ideas. You decide.

Over 100 million people choose Wix to create their website. What are you waiting for? Make yours happen today. It's easy and free. Just go to wix.com and create your stunning website today.

[INTERVIEW CONTINUED]

I think it's this important layer of abstraction that is hard to discuss in a scientific way. It's almost as important — I think the platform is a service layer as it expands and as it changes in its

definition. It's almost as important as something like object-oriented programming where it is this layer of abstraction that helps you so much, but we don't quite have the vocabulary to talk about it in the right way yet, and there's a lot — When we talk about skepticism, your skepticism, I think, maybe the — You're probably more skeptical of the word serverless than the word platform as a service.

[0:19:03.5] CM: Yes, exactly.

[0:19:05.1] JM: But these two things are sort of different polls of a gradient where you're essentially leveraging lower level across the network abstractions and you get to build on top of them. Serverless is little more raw, at least if we're talking about the lambda area of things, and I think this would probably fall under your purview of the less opinionated sort of things, but then you have these more opinionated and perhaps more beloved things like Heroku, or Parse. I know Parse was very dearly loved — Firebase, things like Twilio. Yeah, I don't know —

[0:19:49.7] CM: We have that way of looking at the world, that is kind of like object-oriented programming, but we don't quite have the vocabulary yet. I think that's really a stoof, I do. I think you can follow that analogy in long ways. I'd love to hear you speak a little bit more about that.

[0:20:04.7] JM: I don't have a fully formed belief system on it, but you're the one being interviewed. Let's talk a little bit more. For people who do go all in platform as a service approaches for deploying their app, what general pieces of advice do you have for managing those apps?

[0:20:27.6] CM: Yeah. The thing that I kept repeating at Serverlessconf and that I'll continue to repeat today is that outsourcing doing the work, you never need to even outsource caring about it, or needing to understand it, or needing to think about it. You kind of have to just think about them like your family members. You have to take them as who they are, where they're at and don't try and change them, or you can try and change them if you want, but don't get invested in the idea that they will change. You deal with yourself. You take care of yourself and your users without expecting or relying on them doing anything.

That's kind of liberating in a way, just — I hear this in therapy, where you're talking about your family, they're like, "Not to take them for what they're at and you take care of yourself." I think that's a really healthy way of dealing with the world in general and it's certainly applicable here. Nobody is perfect. No system or platform is perfect, anyone who says anything else, who's trying to sell something.

The only way that you can truly know what to expect of a vendor or of a service is; firsthand, you try it, you see how it goes. Secondhand, you talk to people you trust who have tried it; any metrics that any vendor ever publishes, or just for any bullshit. I think having that sort of realist view of the universe and of your expectations is liberating, it lets you design for a world where everything fails and you think about how to take care of your users.

[0:21:55.1] JM: Going down that track further and talking more about serverless or the lambda sort of things, what are the expectations that we should have for serverless? Because I talked to some people and they think of it as a substrate that we will build higher level frameworks on. We talked to Austen Collins who's building a serverless framework that seems to be — His perspective is, "Okay. We'll build things like off and statefullness on top of lambda.

Other people I have talked to say, "Really, the value of serverless is just thinking about areas of your codebase that you can form-fit to serverless and the serverlessnes will manage the auto-scaling of those parts of your code." There seems to be different perspectives with how this might shakeout or what might be useful about serverless. What do you think are the expectations we should — Since you are on the skeptical side, what do you think are the expectations today and what might they evolve into?

[0:22:58.9] CM: Skeptical, but also, remember, I said in love with it. This is a very much love-hate, I care so much about it, but I get cranky when I see people doing things that I think will undermine the model, 'cause I am fundamentally a believer that love and hate definitely coexist.

I'm more of the first camp, with the expectation that these first few frameworks are going to be terrible and we're really angry at them. We need ways of abstractly talking — Let's be clear. This service model is not for every service, and it is not for every workload, and it is not for everything. I think that it match really, really well to certain types of workloads and not at all to

others. We're talking about something that is has to be very loosely coupled. You'd have to expect that anyone of your vendors can go away and can be mostly okay.

I think that this is a great model for — For example, a company like Facebook could be — In the future, be very large and use this for batch processing. It's on demand compute, background jobs, great for this data warehousing, analytics, that sort of thing. Anything that is — Where the surface is close to the user and they're relying on it right away, immediately. You don't want to distribute that across too many vectors for outages and — You know what I'm saying? Your web app, the thing that your users are interacting with, the fabric of it directly, could only be one thing. The components of it can roll off and be loosely coupled — I feel like I'm going down a hole. Let's rewind a little bit.

[0:24:37.5] JM: Okay.

[0:24:38.6] CM: I'm trying to compress like, "Oh my God! So many rants into brief overviews." Maybe pick out one thing that you would like me talk about and we can go up a little bit.

[0:24:48.9] JM: The vision that I see for serverless, I actually am a little more on the Austen Collins' side of things, where I think that — If you look at serverless from a very fundamental perspective, it is just taking the little bits of computing that is dispersed throughout the AWS datacenters and aggregating them and saying, "Okay. Let's use this," and so you can sell them at a much lower cost.

[0:25:15.6] CM: To put it, I'm saying I also — I'm on the Austen Collins' side of —

[0:25:19.3] JM: Okay. I think much like with EC2, we've got two things. First of all, EC2 early on, there were more outages, there were more bugs. It was very legitimate for people to be a little bit paranoid about putting — Going whole hog into AWS, but now we're at a point where it's more secure to be on AWS in most cases. You can rely on AWS and arguably rely on AWS more than you could rely on —

[0:25:51.1] CM: Not arguably. Absolutely 100%. Yeah.

[0:25:53.9] JM: Absolutely 100%. My sense is that there is just this inevitability that eventually it will be figured out how to make this work for serverless, even though right now we have this amorphous blob of unreliable compute that serverless provides to you. We'll get there eventually.

To get to a question, I guess — I don't even know if I have a question. Do you —

[0:26:17.6] CM: Okay. Liberate. I do not think that "serverless" is inevitable in that sense for everything. I think it is for some things. The step that you care about the most, your fundamental business logics, whatever it is that's different about you, that's compelling about you that nobody else is doing, you're going to have to be very involved in that. You're going to have to be very involved in that. You're not going to be able to outsource. You're going to be able to outsource components. For example, if you're building an app that does — God! What's something an app would do? For us, what we're building —

[0:26:48.9] JM: Upload an image.

[0:26:49.8] CM: Yeah. Okay. That's commoditizable. For us, let's take us an example. We're building a platform for people to do observability, to understand what their application is doing. There are aspects of these that we have to intensely understand and control. What that is — It turns out for us, that includes a storage engine.

Believe me, we did not want to build our own storage engine. I've spent my career telling people not to build their own storage engines. That's true 99.99% of the time, but we understand what we're doing as radically new and ways that require us build a new storage engine.

I think that at that level, what I'm saying is the things that make up whatever you're building are going to be unique. Whatever is different about you, you're going to have to build. For most people, that is not hardware management. For most people, that is not Qs. You use Qs to build the thing that you're building that's new and different in the world. For some people, the people are building Qs that's a service, that is their bread and butter.

Foe people who are building an image management uploading thing, at this point in time, it would have to be pretty different in ways. How is it different? Is it super fast? Super cheap? Does it have cute little icons that you can enrich it with? Whatever you're doing there that is different, you have to own. I think that for serverless stuff, we're going to see increased shedding. Increasingly, people think ruthlessly about what is different about me? What is in my sphere of extreme competency or creativity? Those are the things that you will not shed. Everything that is, you will.

[0:28:34.8] JM: Now, I want to get a discussion of Honeycomb, which is the company that you're working on now. I want to talk a little bit about your experiences at Facebook before we get there, and — I don't know. You can talk about this much, but can you discuss what happened with Parse and why you had to shut it down?

[0:28:53.6] CM: Absolutely. We didn't have to, they want it to, and I understand that from a coldblooded business perspective. It continued to be very successful kind of despite their best efforts. I think that the thing that I would say that I've learned from this is strategic alignment matters. If you're being acquired and you're not being acquihired, your strategic alignment matters. It is everything. A C-level sponsorship also matters, which we never had. In fact many execs were surprised when we were acquired, which is not a great sign.

C-level sponsorship is really important, but strategic alignment — Look at the varying examples of Instagram versus Parse. Instagram and Facebook, so much synergy, right? They're both big social apps went to serve lots of photos and make people more open and connected. Their platforms aligned. Their missions aligned. They were able to just really — Not to say it wasn't hard, but they aligned.

With Parse, it was always more of a, "Well, let's see if we can make developers love us." By "a thing that developers love", there were some ideas of us helping Facebook platform, but they really didn't want to listen to the thing that we had to say about what developers liked and didn't like. You can't really have impact if they don't want to listen. We just weren't aligned. We were not aligned at all, and they kind of kept us on as a cost, you know, a thing that cost a lot of money, but it was a cool project and lots of people loved it, so we kept it on for a couple of years and shunt it around for department to department and we tried a few things to get there to be

overlap, but Facebook basically decided to stop sponsoring other mobile apps other than the Facebook app.

There came a point in time where it was just like, “Why does this thing exist?” Which was too bad, because gross continued to be phenomenal even though there was never really any investment in it, which tells me that there is an appetite for this sort of thing, a huge appetite. We were early. We did well. We maybe didn’t focus on getting revenue. I was always pretty irritated that we stopped caring about revenue after we got to Facebook, because I would say, “I don’t want to be a gift with purchase.”

We’re a liberty capitalist society and worth two people is demonstrated by if they’re willing to pay for it. People were very willing to pay for Parse increasingly so, but we didn’t really worked to make it revenue neutral and that, I’m sure, didn’t help. It was pretty expensive by the end. Why is why with Honeycomb we are, from the very start, we’re focusing on pricing as one of the most important things, and it’s hard. It’s really hard to price these things.

[0:31:39.2] JM: Yeah, I spent a lot of time thinking about pricing of podcast ads and I know pricing is a complex thing. As far as Parse, I think the obvious analogy, I’m sure you thought about is Salesforce acquiring Heroku. In that case, the patient did not reject the organ transplant despite it being somewhat unaligned with the rest of the body. Is there something — Facebook, I think of as more of a — Measurement structure of Facebook is a little — I don’t understand it as well as well the Salesforce model is represented in my head, but I think of a Mark Benioff, his legacy of working at Oracle. He’s probably somebody that is more focused on the revenue and the cost structures of the individual business units rather than the synergies and the overall cost structure of the bigger entity that I think of Facebook, the Facebook mental model. Do you think it’s a failing of Facebook, that Facebook was not able to — Well — Yeah, I don’t know.

[0:32:44.1] CM: I think they should never have acquired us, and I think we should have consented to being acquired by them.

[0:32:48.8] JM: Just a culture problem.

[0:32:51.1] CM: Hmm?

[0:32:51.1] JM: It is a culture problem.

[0:32:52.6] CM: It's that Facebook is a giant web app. That's all we care about. It may seem simplistic and reductive to put it in those terms, but the only things that succeed at Facebook are the things that wrap into this web app. The only reason that they're investing in that VR goggles thing, Oculus, is because they think that's the future for Facebook. The reason Instagram is succeeding is 'cause it's successful for Facebook. Parse was successful at Facebook for as long as they thought that the mobile app ecosystem was important for Facebook. That wasn't long after we got there, honestly.

It's possible that if Parse had been comprised of slightly different individuals, we may have been able to make the argument that somehow we could — I don't know. That's easy to engage in hypothetical, is we had a lot of really strong-willed, strong minded people, a lot of whom quickly quit Parse and went back to Google.

I really do think that there are two type of big company people, there's Facebook type people, and there's Google type people and they don't get along, and there were some impedance mismatch there for sure. We loved our mission of helping app developers. Facebook doesn't really want to help app developers because they want eyes beyond Facebook. For a time, that wasn't true. For a time, they were like, "It can be a hug for blah-blah-blah." I think you can see from the investment caring about — I'm going to say things that are going to get me in trouble, so I'll probably just fade off there.

[0:34:21.2] JM: Okay. To fade you back on, the Facebook versus Google employees in the Valley, I find that interesting contrast. I worked at Amazon in Seattle which maybe is a third example, but perhaps we —

[0:34:33.1] CM: I really liked if Amazon had acquired us, honestly. Go on.

[0:34:36.1] JM: Yeah. That would be an interesting hypothetical to run, but when I think of at least the Amazon innovation model, because I understand that just from my extremely limited time there, eight months, if somebody stands up their own little company thing within Amazon,

you can get buy-in from the management, like, “Okay. Do that thing, and run with it and stand it up. If it starts to gain some traction, my sense is that Amazon will look for a way to — ” Yeah, maybe there are some coincidental synergies, or maybe you have internal competition and maybe one of those things works out. That to me sounds like more of a Google decentralized innovation approach that perhaps does not exist at Facebook?

[0:35:16.2] CM: Yeah. This is why I think that both Google or Amazon paid, Parse would have been wildly successful at. Because both of them are used to nurturing or leaving alone — Just like Salesforce mostly left Heroku alone for a while and let them kind of — It’s a sad but true fact of life that people — The original Heroku team, by and large will not be happy at the Salesforce Heroku, and vice versa, which is a normal maturation of the company thing that goes on.

They kind of needed to give a time to grow up and to like the sorting that it inevitably goes through. I think that all three of those companies are just very different form the Facebook model. They’re looking for other ways of making money, and other ways of innovating, and other ways of being in front. They’re open to nurturing bubbles for lack of a better word. Or little internal bubbles of creativity and enthusiasm.

Whereas with Parse at Facebook, it was, they were jokingly angry at me for — So many people wanted to work at Parse. That’s almost one of the things that they were most irritated by, because they didn’t want that detracting from the beating heart, lifeblood of the company, which is the Facebook web app.

[0:36:28.3] JM: Wow!

[SPONSOR BREAK]

[0:36:37.3] JM: Indeed Prime flips the typical model of job search and makes it easy to apply to multiple jobs and get multiple offers. Indeed Prime simplifies your job search and helps you land that ideal software engineering position. Candidates get immediate exposure to the best tech companies with just one simple application to Indeed Prime.

Companies on Indeed Prime's exclusive platform will message candidates with salary and equity upfront. So if you're an engineer, you just get messaged by these companies and the average software developer gets five employer contacts and an average salary offer of $125,000. If you're an average software developer on this platform, you will get five contacts and that average salary offer of $125,000.

Indeed Prime is a 100% free for candidates. There are no strings attached, and you get a signing bonus when you're hired. You get $2,000 to say thanks for using Indeed Prime, but if you are a Software Engineering Daily listener, you can sign up with indeed.com/sedaily, you can go to that URL, and you will get $5,000 instead. If you go to indeed.com/sedaily, it would support Software Engineering Daily and you would be able to be eligible for that $5,000 bonus instead of the normal $2,000 bonus on Indeed Prime.

Thanks to Indeed Prime for being a new sponsor of Software Engineering Daily and for representing a new way to get hired as an engineer and have a little more leverage, a little more optionality, and a little more ease of use.

[INTERVIEW CONTINUED]

[0:38:25.3] JM: Interesting. Getting away from the Parse experience, so at some point you moved to solving production issues, I guess. Maybe when Parse was shut down, you moved into more of an operations role. Is that accurate?

[0:38:40.6] CM: No. God no.

[0:38:42.3] JM: Okay.

[0:38:44.0] CM: I left Facebook, basically, a year ago, August. Wait — Two Augusts. But I didn't leave-leave for — I took about six months to figure out if I was going to come back and join the MySQL team or something else.

[0:38:57.3] JM: So you went on sabbatical.

[0:38:58.3] CM: Yeah, basically. I had a bunch of conferences to speak at and stuff, so I kinda have to falloff just kinda wandering the world and thinking about the future. They announced that they were shutting down Parse five or six months after I left Parse.

[0:39:11.2] JM: Oh, I see. I read *Chaos Monkeys*, I don't know if you read it, but it's a book about a guy who gets his company acquired by Facebook and then he leaves Facebook eventually.

[0:39:21.8] CM: No. It's been one of those things I've been looking at and there's been like little too painful close to home that I haven't read it. Is it good?

[0:39:28.6] JM: Oh, wow! It is unbelievably good. I have been trying to get the author on this show, but he's done a bunch of good podcasts if you want a free sample of his thinking and his philosophies. He talks about this interesting — He gets fired eventually from Facebook, but he talks about there is a culture in Facebook that's really fun to be at and it's really addictive and when you leave Facebook, it sounds like it can be somewhat of a traumatic experience. I think it's a case —

[0:39:55.8] CM: Leaving Facebook?

[0:39:57.1] JM: Yeah, leaving Facebook.

[0:39:58.3] CM: Oh, I think it is for a lot of people.

[0:40:00.1] JM: Was it for you?

[0:40:00.8] CM: No.

[0:40:02.4] JM: Okay.

[0:40:03.6] CM: No.

[0:40:04.9] JM: Why do you think that is? Difference in your personality?

[0:40:07.8] CM: I don't know. There's something about being non-consensually acquired. I mean, I cared about Parse massively, or I wouldn't have stayed. I turned down Facebook years ago, literally, 2008, 2009 because it was too big for me.

[0:40:26.5] JM: It was funny when Vine got shut down, the tweets from the Vine founders, they were just like, "Never sell your company."

[0:40:33.0] CM: You know, it's not that. It's just like — You're entering into a relationship. I think it's really hard for employees. For founders, at least you consented. It was your cost benefit analysis, your — As a founder, I know this, hugely now, very, very intimately that the cost are always weighing on you and it's just a tradeoff of these set of costs with that set of costs. I totally understand why the founder sold it to Facebook. For the rest of us hearing it was a shock. It was not what we signed up for. Let's just start with the easy thing, like three hours of commuting a day from the city. Let's just start there and then go to all the other bullshit around it. It was a labor of love and we lost a lot of people right after we joined because of that, because it wasn't what they signed up for.

[0:41:21.9] JM: If I recall around that — Gosh! We're getting right back into it. But around that time. It was funny 'cause Facebook was kinda struggling to figure out what it was. I think this was before the mobile switch and all that added revenue from the mobile movement. It was like, "Maybe we're a platform company like AWS sort of. Maybe we're a social fabric for apps," or something.

[0:41:40.3] CM: Yeah. In that world, it could have been amazing.

[0:41:44.8] JM: Yes.

[0:41:45.4] CM: Yeah.

[0:41:46.9] JM: Okay. Let's get into operations more generally. It is certainly fun to talk about gossip and company culture and what not, but there are plenty of other podcasts that do that better than I can.

[0:41:58.2] CM: I don't usually talk about this. I don't talk about this publicly for obvious reasons, but yeah.

[0:42:04.5] JM: I know you don't. First time for everything.

Totally shifting the conversation, we are in this time when operations are changing, observability is changing, the role of engineers are changing. You suggest that even though we have all these nice platforms that theoretically should reduce our need for operations people, we actually need —

[0:42:26.8] CM: They should

[0:42:27.7] JM: Yes, they should. We actually need operations expertise more than ever. Why is that?

[0:42:33.6] CM: Because operations is just — It's how you run your stuff. It's no more complicated than that. We can argue about the kind of operations that we need, and that's changing a lot. I don't think of operations as being — I'm totally privileged by the amazing people I've gotten to work with. I never — There's like a stigma attached to operations in a lot of places, and I think that's stupid, because when you want to make something awesome or better, you don't usually start by just talking shit about it. You value it. You value it and you talk about what you want to see and how you want to improve. You don't just start going, "We're not going to have any of these." That's just — Come on.

I think that the right way to think about operations in the future is it's like a social contract, everyone participates in it; from the CEO down to support. Every engineer is intimately involved in making it a high quality thing. It's like saying that maintenance is not part of the job of anyone who works at a car company. Everyone should be involved in thinking about safety, security, maintenance, longevity, understanding what you've built. It should be imbedded into your culture and into your product at every level.

Obviously, a car company shouldn't be aiming to have cars need to go to the shop once a month, that shouldn't just respond to any key that opens it. It should be part of your culture, a part of your fabric. That means not devaluing it.

I think that, certainly, the level at which you should all be thinking about it and having to care about it is changing, again, hardware, not our job anymore. Awesome. That freeze up our brain cycles to think about other things. I think that, increasingly, operations is the job of software engineers, and you have "operations engineers", specialists to help you figure it out.

I love the model of where you have — And I've seen this a few times lately and it makes me so excited; 100 people in engineering; 70, 80 software engineers; maybe 3, 4, 5, ops engineers; 3, 4 DB engineers; 3 whatever to it. Those people aren't on call. They are the specialists who help you figure out. They may go around to a new team every couple of months, do some rotations with you, help you bring your observability, your instrumentation up to snuff, help standardize it across the company. Then they bring this — When you have an ops person who's been doing security rounds and doing observability rounds. Then, they can also help standardize and bring a kind of fluidity to every other team to the company, but it's almost embedded consultancy. I love this model. I think we're going to see more of it.

[0:45:29.2] JM: Monitoring has become its own application stack. A company is typically using a collection of mongering tools that's — I remember one company I went to four or five years ago was — It's just like Nagios and not much else. Today, you've got like 5 to 10 monitoring tools. Maybe you're paying a subscription for some of them. Maybe you're on premium for some of them. What are the requirements for a monitoring stack today?

[0:45:58.3] CM: Yeah, that question is gotten harder and more complicated as you've gotten so many other — So many alternatives and options. So many different ways of looking — I always think of the metaphor of the elephant, where everyone's touching a different piece of the elephant and things of it is being a radically different beast. The elephant is just — How do you understand your systems? How do you ask questions of your systems? How do you decide what actions to take to keep quality and performance and all of these things within acceptable parameters?

The answer to the question is going to depend on what your requirements are. A web company is going to have super different requirements from a health group company where everything exists behind closed doors. It's kind of they're doing batch processing. They don't even have any events exposed to the world necessarily. It's much more custom.

I think that there are three critical components, and you can see them reflected in kind of the three pillars today, which are the Datadog-Splunk and the New Relic. You've got metrics and monitoring. You've got events, and you've got APM, application performance management. Here's where I'll plug Honeycomb a little bit. This is where we're thinking about the world that we stand all three. I think that this is important for a few reasons, but I'll talk about the three first.

The Datadog step is your very traditional metrics and monitoring. It's like what we think when we see monitoring. It's very drawing on traditions of Nagios and RRD, the Robin Database thing, where we have rollups and a time interval. You don't have events, they are like kicks and counters. StatsD fits into this really well, but you've got dashboards, and they're fixed, and you don't really interact with them. You scroll and you create dashboards to try and explain. I always joke about how you have always have dashboards to explain the last outage. It proactively notifies you when one of your KPIs or your metrics has fallen out of bounds.

This is a very reactive model. It's the model that most ops people are used to working in, and I think it's a little unfortunate. You're not used to asking questions of your system. You're used to predicting how the system is going to break and watching for that to happen. I'm a little bit down in it, but it's mostly me being contrary, 'cause there's a huge amount of value there.

[0:48:18.0] JM: Sure.

[0:48:18.5] CM: But I think that it's reached a kind of — Reached a point of diminishing returns for sure. Signal Effects is very much in that category. I think that those two are the last and best versions of RRD that we're ever going to see in the world.

Then you've got the APM stuff, which is kind of new. The idea that software engineers should have to think and care about how their stuff is performing. Most of this traditionally took place

offline. You'd use GDB or Purify or something to see if you had memory waves or if your performance had degraded with your unit tests.

In the last few years, there are these shocking new ideas coming up that software engineers should have both power, control, and some, perhaps, responsibility over how their stuff runs in production. I think this is awesome. New Relic is obviously the leader. I think they really were smart to get out in front with a frictionless, you do a gem install, and you got your stuff. It pre-populates. It pre-assembles a bunch of really useful graphs.

The tradeoff there is, again, you've kinda run into a wall. If you want to deviate from the golden path, you kind of can't. This was the right first thing to do, because this is the easiest. There are a very few barriers — Entry could do a lot right off the bad. You can solve a lot of low-hanging immediately, you're hooked. That's awesome. I think that what you're going to see over the next couple of years, or you're already starting to see it, is people get frustrated with that wall that they hit. They can't ask any questions.

The third one is kind of this catch all — Literally, catch all category of log aggregation. This is now messy and far-flung as you would imagine from something that's dealing with all of these strings. Splunk is the leader, but you've AppDynamics, and Sumo Logic, and all these small and large companies that do nothing but bring together your logs and try and help you tease and signal out of it.

[0:50:11.5] JM: Yeah, it's a tough problem.

[0:50:13.0] CM: It is. It's really tough.

[0:50:15.0] JM: What we had — What's his name? The CTO of Sumo Logic, we had an entire show about how that system works. It's like a big problem. You're talking about these disparate solutions that cover monitoring, and you described Honeycomb as what you do when your monitoring ends. I've also heard you describe it as a tool that is useful even for very — Or maybe for experienced engineering organizations who have automated — They've automated everything that they can. They're abiding by the best practices, but Honeycomb solves something else. It does something additive. What is it doing that is additive on top of these

companies that are well-healed, they are intelligently automated. What is it adding to these companies?

[0:51:04.0] CM: Honeycomb lets you ask new questions and ask them really fast and interactively and iterate on them.

[0:51:10.2] JM: About your whole system.

[0:51:11.1] CM: About your entire system, from the IoT mobile devices, through the code you've written yourself, through databases and all of the other systems that you are — Vendor software that you're using to back your system. This is unique. We have to do a little bit of work to get your information in, but we speak JSON lingua franca.

There's a lot of value here, and I would start that feeling of hitting a wall, that feeling of you couldn't anticipate it, you can't monitor for it. Feeling of what do I choose what to index on, I can't predict what's going to break. Those are the questions that we answer.

I think that we were really fortunate to have been at Parse when we were. We kind of got to see into the future a little bit, and being at Facebook, that confluence of things is just so powerful, because when I left, I wasn't planning to start a company, and I hate monitoring. It's like God's cruel joke that I'm doing this. It didn't exist. I just can't even imagine living without it. I believe that the world is hardly even towards feeling the same way.

At the core of what we do is we incentive you to gather as much — If you think of the width of data, we incentivize you to capture every scrap, and you can sample vertically if you need to for cost or whatever. We take events, arbitrarily wide events of key-value pairs, and we aggregate on all of them immediately. There's no indexes. Indexes, again, are another way of predicting what's going to break, predicting what you're going to want to be able to search on efficiently. If you're not indexing it in the log aggregators, you're just doing it on distributed ground, anyway.

We index — We don't even index. We aggregate on everything so you can search on key-value pairs as soon as they enter your system. This allows you to do exploratory debugging — Before, I mentioned of the questions that Parse would have, where some answers that you might

discover would be like apps writes in, they say they're down, and you start going, "Okay. I'm going to slice and dice by their app ID. Okay, now I slice and dice by endpoints. Which endpoints are slow? Okay. It seems maybe the endpoints that are these half dozen which are the right endpoints. Okay, why too slow for this application? Which endpoints are they, or which backends are they going to? It seems to be going to MySQL. Half of those are slow, and the writes to go to MongoDB are slow but the readings are not.

Well, let's take MongoDB first. Are any other apps slow? No, it seems to be just this one. What's the cardinality of that dataset? The Mongo writes appear to be slow, because — Okay. Other apps on that server are slow too when they're doing writes, but maybe they aren't surfacing or they haven't been complained about yet. Okay, it seemed like the write log percentage on that MongoDB primary spiked about 5 minutes ago when this other app committed an update that started saturating the write lock on this particular database and it's holding this lock. There's contingent —"

That's one example. Another example would be an apps says that some of their users are saying it'd down, and it turns out to be only their users who are coming from France who are using iPhone of this version, who are using iOS whatever who are running this version of their application, who are going to this particular router, who are — It's just, I feel like in a mature complex system, there are no easy problems, because you've automated them away, they're solving themselves. They've being remediated. There are only hard new questions and there is nothing out there that helps you ask new questions.

You can imagine the old sub getting paged as you look at your phone and you're like, "Oh, I know what that is. I'll go remediate it. I know what this is." In the new world, every time you get paged, it should be new and baffling and kind of terrifying. The hard thing about hard things is nobody's done them before. You don't know how to do it. What we do really, really well is we don't walk you into trying to predict it. We help you answer arbitrarily hard questions really fast.

[0:55:20.1] JM: You're describing this system — Honeycomb is to your systems like an IDE is to your code.

[0:55:24.8] CM: Yeah, I've used that metaphor before. Yes. It's exactly like that.

[0:55:28.1] JM: Yeah. It's on your website, I think.

[0:55:31.4] CM: Thanks.

[0:55:33.8] JM: It makes complete sense. You've got — Because you need an IDE as an assistant to wrangle these complex systems, that helps you a lot in wrangling these complex systems.

[0:55:42.6] CM: Yeah, you can't hold it on your head anymore. You can't. You shouldn't be expected to. We've relying on senior engineers do this, to hold the systems in their head. I know 'cause I've been a senior engineer. I was on my honeymoon, I'm on the beach, I don't have my computer, and they're calling me. They're like, "We're so sorry, but Parse have been down for an hour and we can't figure it out."

I'm just like — I just start asking questions, and I debug by smell more than anything else. I just sense it, and I sense it because I built, 'cause I know it. This feels really good. It feels really good to be that super hero engineer, but it doesn't scale. It's increasing the abilities of event he most senior engineers. Facebook still relies in their senior engineers who have been around for 10 years who can smell this stuff. Does this feel — it's hard to bring new people in. We can't debug by intuition. We have to bring it into the realm of data.

[0:56:30.4] JM: Yeah, you don't want the situation where every time you've got some unheard of bug, the senior engineer has to open eight terminal windows and flip between them and —

[0:56:40.3] CM: Or being called back from vacation or — The heroics era needs to end. Yeah.

[0:56:45.6] JM: Yeah, the heroics era and the era of, "Oh, you're piping eight Unix commands," and, "What are those eight Unix commands?" He's like, "Don't worry about it. You don't need to know."

[0:56:57.8] CM: But the other component of this though — I'll talk about before we move along, is those senior engineers have some much value to add, it just has to stop being them. Which is

why the first thing that we're — People always ask us, "Oh, are you doing predictive analytics? Are you doing anomaly detection and all these stuff?" I'm like — I internally roll my eyes because everybody claims that they're doing it it's kind of smoking crack. Sure, we can do it as much as they're doing it, but I don't think that's just powerful here. I think what's powerful at this point is your team.

If I told you could have your teammate's idea of what you should be looking at or what my computer's idea of what you should be looking at, which would you choose? You'd be dumb not to choose what your teammate thinks you should be looking at.

What we're doing is we store the history of what you've been searching for and we're encouraging you to add comments and explain back to yourself what you're seeing, because debugging is a social activity. It just is. Even if it's just and your past self and your future self, you have to add little cookie crumbs for yourself to explain what you were looking at yesterday, 'cause you're going to forget today. You have to add little nudges for your future self.

On a team, you really want to be interacting your data. You want to — I think of this in terms a lot of on-call handover. We have historically spent a lot of time post-morteming and doing a hand-of and stuff, when, actually, just what I want to be able to see is, "Oh, I have a problem with Cassandra." "Didn't Christine have a problem with Cassandra last week?" I want to just go back and look at what she did and the note she left for herself that explained what was happening, and then I'll pull that into my own and I'll tweak it.

The goal here is to kind of not start at zero when you're debugging, but start by jumping to 85, or 90, or 95% of the way to the solution. By building up this knowledge base of the problems that you guys have seen, it's like you're — It's like the heat map — It's kind of like an emergent production run book where you're relying on yourselves to write down what's happened, but you see what each of you actually ran when you were trying to explain stuff. You curate for each other, even curate outage or like post-mortem reports. It's almost like a playlist of what each of you — Again, on the elephant, what each of you has seen that made up this complex event.

The other thing about outages in the modern world is that there never one thing. They're not even two things. It's usually this black swan event of 1, 2, 3, 4, 5 things I'll hit at once, which is

why they're so hard to reconstruct. It's so hard to test for, and that's what we do so well, and it really requires drying on all of the different silos of deep knowledge across your entire organization.

At Parse, often, it would be like a combination of what happened on the SDK, and they rolled a new SDK in the API team, and the backend, and the database. We all had to be there to figure out what was going wrong.

[1:00:00.0] JM: Yeah. I think it's a great time to build this tool, because we're getting better at building these rich user interfaces, the technology is getting better for building them, and I can imagine this sort of developer tool that — I mean I haven't used Honeycomb. I've taken a look at it. I've seen what people have to say, but it sounds like it is a very rich interface where there's a lot of surface area that you cover already. The onboarding process is not too tough, and it's — There's plenty of room for you to expand, I'm sure.

I would love to talk more about Honeycomb, but I know we're running up against time and I kinda want to just — As we begin to close off, ask you a question that's been a theme in some recent episodes is the question of how you build a developer tools business, 'cause you were able — Kinda were able — You did this at Parse. The business sort of sold before it was — I don't know if you were profitable when you sold, but —

[1:00:56.2] CM: No. God.

[1:00:57.5] JM: No. Okay. No. Not profitable. Anyway, the developer tools business is interesting. Do you build an oper source? Do you not build an open source? What are the — Just to close off. What are the lessons that you've learned about building developer tools businesses successfully?:

[1:01:13.8] CM: Oh my God! Yeah, in 30 seconds or less, how do you build a business? I think that, for us, the engineering is the easy part. It's just like we're fortunate to having amazing engineers here, but everything else is really hard, which is why I've spent all my time thinking about it. The pricing from day one, we're agonizing about pricing, and it's getting easier, believe me. It's getting harder.

It's also very — I'll just say, especially with recent electoral events, the world is always unpredictable and it's become even more so. I'm really cynical about the valet promises that you shouldn't need to think about revenues, that it will come after the users do. I know from running infrastructure services that they are very costly. I hope the world is willing to pay for them. I see evidence that they are.

The thing that happened with mail 10, 15 years ago, where Gmail appeared on the scene and we all suddenly went, "Huh! We're all spending core engineering cycles treating spam filters and antivirus and running IMAP servers, maybe we should not do that. We suddenly freed up engineers, right?

I think that that's starting to happen across the industry. I think it's still really hard to surface to being pushers — being keepers, bookkeepers, the massive underlying human costs. Especially when some of these services are not super well baked, and still incurs human costs on your side. Gmail was not super reliable in the beginning. Parse was not super reliable in the beginning, and I imagine that there were times when it cost more engineering cycles than they've saved you, but that trend is curving in the direction really fast, and I would say that it's worth your time to start investing and knowing how to do these serverless things well.

It is a special skill. You're not just pushing the need to care about that off. You are changing and hopefully shrinking the amount of attention that it does take. I think worth investing in that for everyone today. I think that for those of us on the production side, remembering to always bring the conversation back to value. What is the long term value that we are bringing in your business? I think that that argument is pretty unsalable at this point, and so we both have to think about ourselves in terms of sustainability.

Maybe growing a little bit slower, but growing more sustainably, and bringing the conversation back to value, we are preventing great value. We're doing it better than you can do it yourself. You're investing in partnering with us and it's going to pay off. I think that that's the right approach, whether it's open source or not. I think, generally, a combination of hybrid approaches where I wouldn't want to run something on the client side that I couldn't see and change if I had to. I understand the needs and I have experienced needs of a pure open source business model

as well. It's really rough. Close sourcing the server side and open source on the client side is what we're doing, and I think it's the best model that I've seen overall.

[1:04:14.0] JM: Charity Majors, thank you so much for coming on Software Engineering Daily.

[1:04:17.0] CM: Thank you for having me. This has been really awesome.

[END OF INTERVIEW]

[1:04:25.1] JM: A few things before we go. If you like the music on Software Engineering Daily, you might like most recent album called *Commodity*, which features a lot of the songs that air on this podcast. My artist's name is The Prion on Spotify, Apple Music, and Amazon.

Also, Software Engineering Daily is having our second meet up, March 9th at Galvanize. You can find details on our meet up page. Finally, we are about to post our second listener survey, which is available on softwareengineeringdaily.com. This is your opportunity to have your voice heard and let us know how we can improve.

This data is super valuable to us and we'd look at every single response, so please take the listener survey at softwareengineeringdaily.com. Thanks for listening to all these announcements. We'll see you next time on Software Engineering Daily.

[END]

