EPISODE 835

[INTRODUCTION]

[00:00:00] JM: Google's codebase is managed in a single monolithic repository. An engineer at Google can explore almost any area of the codebase within the entire company. In order to enable this, Google has built tooling to support the gigantic monolithic repo, including a virtual file system and a set of build tools.

A monolithic repository is not to be confused with a monolithic deployment. Google's infrastructure is not a deployed monolith. It consists of thousands of small services interacting over a network and scaling individually, but all of the code for each of these different independent modules is in the same version control system.

Ciera Jaspan is a staff software engineer at Google working on developer infrastructure. She worked on an internal research project within Google to find out how engineers felt about the monolithic repository system and how it compared to a large number of small repositories, which is an alternative form of management.

Ciera joins the show to discuss repository management, internal tooling and Google's approach to researching developer productivity within the company.

FindCollabs is the new company I'm building. FindCollabs is a place to find projects and build them with collaborators or to start your own projects. We have a hackathon. It's the FindCollabs Open, and you can go to findcollabs.com/open and compete for the $2,500 in prizes. We've got prizes for projects around machine learning, music, visual art, React.js, podcasting, cryptocurrency, computer games, all kinds of stuff. If you're looking to build projects, we'd love to see your projects on FindCollabs. You can go to findcollabs.com/open and check it out, and I hope to see you there.

[SPONSOR MESSAGE]

[00:02:03] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit. After you're in the Triplebyte system, you stay there, saving you tons of time and energy.

We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. So take the quiz yourself anytime even just for fun at triplebyte.com/sedaily. It's free for engineers, and as you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. So Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that's really useful to engineers. So check out Triplebyte. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte, byte as in 8 bits.

Thanks to Triplebyte, and check it out.

[INTERVIEW]

[00:03:52] JM: Ciera Jaspin, you are a staff software engineer at Google working on developer infrastructure. Welcome to Software Engineering Daily.

[00:03:59] CJ: Thanks. It's nice to be here.

[00:04:01] JM: The topic of our conversation today is the distinguishing characteristics of monolithic repositories versus alternatives. Could you start us off by defining what a monolithic repository is?

[00:04:16] CJ: Yeah, sure. So we defined a monolithic repository as a source repository that has five properties. So one of them is it centralized all of your source code is in one location as supposed to in separate smaller locations. It's also visible to all of the engineers working on all parts of the codebase. It's also synchronized. What we mean by synchronized is that the repository is commits to the [inaudible 00:04:42] trunk base. There's no branches. Everybody is always committing to head. One of the more unique things about a mono repo is that they are complete.

So a mono repository has all the dependencies for all your binaries all checked in, and more interestingly, they are unversioned. So you don't have multiple versions of a particular dependency. Everyone is using the same version and it's always the one that's currently at head. The last part of this is that a mono repo has a lot of standardization. In order to make these all work, you have to have shared tooling. So a single build system, for example, that everybody is using.

[00:05:19] JM: If I have not worked at a large tech company, I may not be aware that there is any alternative to this monolithic repository. When I was in school, if I was working on a project, I would just have one GitHub repo. That repo might have feature branches and other kinds of branches. But I would never have separate repos for the same project. What is the alternative to this monolithic repo strategy?

[00:05:49] CJ: The alternative would be that every sub-project, every project has their own repository, and a lot of companies do do this, where they're going to have for every piece of a project in a large company, they'll have their own repository and the teams then can only view and access their own repo, but they can't access the repo necessarily of all the projects at the company.

[00:06:09] JM: By the way, we're talking exclusively about git repositories or are we talking more generally about just software repositories?

[00:06:17] CJ: No, more generally. So this can be, of course, git. You can use git as a mono repo. You can have lots of small git repositories. But this could be any source version control system.

[00:06:26] JM: What are the problems of a monolithic repository, because it sounds great to just have one centralized place for my code.

[00:06:35] CJ: Yeah, there are some problems here. So one of the big ones is that, quite frankly, at some point you run into scaling issues, and this isn't as much of a problem [inaudible 00:06:46] example if you're a student and you've got some relatively small amounts of code all in your one GitHub repo. But when you're at the size of Google's codebase, we've got – What was it? 2 billion lines of code. So a lot of tooling won't actually scale up to that size. So you have to then have a lot of developer tools around just specifically for that type of repo.

[00:07:10] JM: What kinds of tools would you need? Sorry to interrupt you.

[00:07:13] CJ: Sure. Yeah. So one thing you need for example is a virtual file system, because you literally can't have the entire repository on your machine. Some of the other tools that we've built up custom within Google for this are having a common build tool. Actually, our build tool, Bazel, is open source. Other people can use it as well.

In order to navigate all of these code, you need to have a common code indexing tool, because, again, Eclipse and your typical IDEs just simply can't navigate that much code. At Google, we use an indexer that we've built in-house called Kythe. This is also open source at this point so other people can use it as well.

[00:07:54] JM: The virtual file system for my gigantic repository, it almost sounds like a request for startups. Why doesn't that exist, or are there virtual file systems that people use more broadly? Because I've heard about this phenomenon where Google has this amazing virtual file system for managing their repositories, but I have not heard of that being used anywhere else.

[00:08:16] CJ: That's a really good question. I don't know why. This might just be a matter if it's really hard a startup to get involved into a developer tool space. You don't see that many companies being successful there.

[00:08:27] JM: Well, except in Israel. Although, also, I think people don't necessarily want monolithic repos, and I'm not quite sure why.

[00:08:36] CJ: Well, there are some good reasons not to want them. One of the common complaints we heard before going into this study was that people found the Google codebase very daunting and complex, especially new engineers to the code base, and it is scary. Your first week when you open up your code browser and you've realized how many files there are and you're going, "I don't even know how to navigate this. How do I start? How do I find an example?" It can be a little scary and daunting to understand that, and it takes a little while to understand how to use the tools to get around that.

[00:09:08] JM: On the other hand, if you're only exposed to some small subset of the codebase, you may feel like you can't get a bigger picture. I dealt with this when I was an engineer at Amazon. There are definitely virtues to only being exposed to some small subset of the codebase, and I think I actually could have accessed the rest of the codebase if I wanted to to a large extent. But I kind of liked the default, like you can see any part of the code and are encouraged to see any part of the code.

[00:09:37] CJ: Yeah, and that turned out to actually be in our study though. One of the major benefits that engineers said about this codebase is that you get this visibility, and that enables a huge number of useful things. So you get code reuse benefits, being able to find the code that you want to reuse. You can also easily find examples, and this is where tools like code indexers, like Kythe, come in handy. I can go and find a library that I want to use and I can say, "Well, how else is everyone else at this company using this library? What are some examples?"

The other major benefit we get, and this comes not even from being able to see the whole codebase, but also the ability to commit to the whole codebase, is we receive updates from the library owners. So I as a software engineer have many dependencies within my code, and when those libraries update to a new version, they actually send me changes to update my client to the latest version.

[00:10:34] JM: People listening to this who are frequent listeners, who are longtime denizens of the software world are going to be thinking at this point, "Okay. Wait a second. So Google runs

this big monolithic repository. I thought I've been hearing for years that this is like a microservices-based organization and people are like pushing code independently, and this is like merging with the codebase, and then it's like getting deployed in its own small microservice, and then that microservice is getting Canaried out independently. It's getting slowly rolled out." Now you're telling me that actually everything is in one big monolithic repository. Help me understand that distinguishment.

[00:11:19] CJ: Yeah, this is distinguishing feature between a monolithic repository and a monolithic architecture. So Google has a monolithic repository, but we do have a microservices architecture. On the other hand, you could have the opposite. You could have lots of small repositories, but when they're built up into a binary and you release it, it's one monolithic architecture. We don't do that. We do have a mono repo. All these microservices are stored in one place in the code, but we release them all independently.

[00:11:49] JM: That is a really important clarifying point, and I think it's so important because people get the sense that if you're doing things monolithically in terms of repository management, you must therefore be doing things monolithically in terms of your deployments, and that's simply not a binding truth. So why has Google arrived at this way of managing repositories versus managing deployments?

[00:12:14] CJ: You know, that's a good question. I don't know the history of how we ended up here. I suspect it rather happened organically though.

[00:12:23] JM: Yeah. The context for this conversation is that you have done research and study within Google around repository management. Can you set some context for this conversation and how you gathered data and how you started to work on this project of researching developer preferences around repository management?

[00:12:46] CJ: Yeah. So I'm on a team here at Google that sits in the developer tools groups. We're called the engineering productivity research team. Our goal is to try to improve the productivity of all engineers at Google through making improvements to the developer tools. So we're always looking at how can we better improve these tools and where is kind of the best

place for us to put our resources so that we improve, make the most impact across all of Google engineering.

Part of that also though means understanding and questioning the status quo and understanding where we're at. We don't want to end up in a case where we're in some sort of local maximum and we can't improve farther. So we need to understand why we are currently at the place we're at.

[00:13:33] JM: Outline for me the spec for your research process. How did you architect your workflow before you got into it?

[00:13:44] CJ: So our research process is we kind of gather a lot of different types of questions people have about both our source repository, but also our software process. We look for where the biggest questions, where the things that are making the most impact for developers. Repositories are of course the first place you go to. So that's the place we have large impact.

We then formulate some research questions and hypothesis around that based upon sometimes anecdotal evidence, or based upon prior studies that we've done, and then outline [inaudible 00:14:15]. Then from there we outline actually a study plan and go ahead and execute the study. We use a lot of both qualitative and quantitative methods usually combined. The study was a good example of that, where we have a survey and then we back it up with the logs and LSS.

[00:14:31] JM: If Google has this monolithic repository, how did you get counter examples from within the company?

[00:14:39] CJ: So what we did is we asked engineers about the prior experiences. So for this study, one of the things we had done was a survey. We surveyed – I think we had over 860 responses, and of those engineers, over 300 had prior experience working at a company where they had used multiple repositories, and another 300 had also previously had experience working in open source. So we are able to narrow in on just those engineers who could do a little bit of compare and contrast.

[00:15:11] JM: Now, all due respect to that data gathering process, because I think this is just an area where it's like a hard to really get the necessary population sizes to have like a bulletproof dataset. But do you wonder like if Google – Maybe doesn't even matter. Like maybe if Google was to – And I guess we can't know in the scope of this conversation, but if Google was to have ended up in a micro repo kind of situation, perhaps it would have just built like good enough tools to have a great micro repo experience.

[00:15:44] CJ: And we might have. It's hard to say. I think though, one of the things we found from the study though is we were trying to understand what are the benefits of each type of repository and what are the tradeoffs between them. So one of the things we did find was that tools are really important, but people love their open source tools as well. GitHub came out as – Or git in general came out as a tool that people hugely love and they will pick a repo simply for git in face.

However, there were other tradeoffs we were seeing between multi and monolithic repositories around the tradeoffs, around how you manage your dependencies and whether your code has a lot of consistency across all your codebase, or whether you have flexibility to make unique development choices for your project.

[00:16:32] JM: So do you think it might be the case that unless we have the proper tooling, we might have to do micro repose in many cases. If we build up this necessary tool and like the virtual file system and a great build tool system and we actually implement that and we have standardization around it, perhaps we can get to monolithic repos. But for lack of tools, perhaps we need micro repos.

[00:16:57] CJ: Sometimes you definitely do. Micro repos to provide benefits along the lines of, for example, security. You might be working within an environment where you can't trust people across the company or maybe you have people who are third-party contractors working in your code base and you want to limit them to one part and not another part. So there are definitely good reasons for using multi repos.

[SPONSOR MESSAGE]

[00:17:28] JM: When a rider calls a car using a ridesharing service, there are hundreds of backend services involved in fulfilling that request. Distributed tracing allows the developers at the ridesharing company to see how requests travel through all the stages of the network. From the frontend layer, to the application middleware, to the backend core data services, distributed tracing can be used to understand how long a complex request is taking at each of these stages so the developers can debug their complex application and improve performance issues.

LightStep is a company built around distributed tracing and modern observability. LightStep answers questions and diagnosis anomalies in mobile applications, monoliths and microservices. At lightstep.com/sedaily, you can get started with LightStep tracing and get a free t-shirt. This comfortable, well-fitting t-shirt says, “Distributed tracing is fun,” which is a quote that you may find yourself saying once you are improving the latency of your multi-service requests.

LightStep allows you to analyze every transaction that your users engage in. You can measure performance where it matters and you can find the root cause of your problems. LightStep was founded by Ben Sigleman, who is a previous guest on Software Engineering Daily. In that show he talked about his early development of distributed tracing at Google. I recommend going back and giving that episode a listen if you haven’t heard it. If you want to try distributed tracing for free, you can use LightStep and get a free t-shirt. Go to lightstep.com/sedaily. Companies such as Lyft, Twilio and GitHub all use LightStep to observe their systems and improve their product quality.

Thanks to LightStep for being a sponsor of Software Engineering Daily, and you can support the show by going to lightstep.com/sedaily.

[INTERVIEW CONTINUED]

[00:19:39] JM: What was the outcome of this study? What data did you find after surveying this large swath of developers who had experience within Google and outside of Google?

[00:19:51] CJ: Yeah. What we found was there was a lot of really interesting tradeoffs between monolithic and multi repo source, multiple repository systems. We kind of themed this around five areas. So we did see, as we were expecting, that visibility became a major reason why

people prefer monolithic repository, and not just visibility, but all the benefits that it provides for them. We saw that developer tools, as I had mentioned, is another major reason why people might choose one versus the other, which really has nothing necessarily to do with mono versus multi repos, but people do have a strong associations to their tools.

More interesting things I thought were around the two big tradeoffs between these. So one of them was on consistency versus flexibility. So people identified that in a mono repo, you have forced consistency. For example, at Google, we have one style guide for each language. In fact, there's even only one version of the language. We're all on the same version of Java right now, for example, and you don't get a choice in that. You can end up coding in the morning and you go to lunch and you come back and, well, now, you're on Java 8, because we all migrated together. But there's some consistency in that that people like being able to always know that they can build everyone else's code and they can always read and see everyone else's code.

Another thing with the multi repo though, you have a lot of flexibility. You have flexibility to select your own programming language that you want to use, to pick your own development stack, to choose your own tools that might be different from your neighboring project.

[00:21:26] JM: And if we had free reign within an organization to choose our own workflows, to choose our own tools, how do you look at the tradeoffs there? Is that strictly a good thing or do you ever have a problem where there's too much heterogeneity of different workflows within an organization?

[00:21:45] CJ: I think especially for Google, having that much heterogeneity would cause us to buckle under. We just have too many – Too large of codebase to be able to support 50 different programming languages, for example. So we've got to limit to which programming languages we use in our production code. As I said, even which versions we use.

So at least for a large company, I think there're a lot of benefits to consistency, and we actually did see that in our data as well. While engineers at Google noticed this tradeoff between consistency and flexibility, when it came down to choosing which do you prefer a mono repo versus multi repo, people would pick a mono repo for the reason of consistency. They would never pick a multi repo for a reason of flexibility, which I found kind of interesting.

[00:22:33] JM: As you were planning out and starting to do this study, did you find that people actually had a debate in terms of their preferences? Because I'm thinking about this and I'm like, "If I had all the build tools necessary to have this kind of situation, I can't see wanting a micro repo situation at all." I mean, I guess the security benefit, that seems like the one area that seems kind of useful. But was there anybody in Google who was actually like, "I think micro repos are going to be favored as they were like predicting the results."

[00:23:10] CJ: Yeah. So one of the things we saw was the, well, a majority of Googlers did prefer using a monolithic repository. There was a minority that prefers multiple repository systems, and their reason, their primary reason we are seeing there what that people like the stability of the dependencies. This is other tradeoff that we were seeing, where in a mono repo, you're always using the version of your dependencies that are currently at head, which means as soon as somebody makes a change to that, you're on it. If they break it, they're breaking you too. So there's always a chance that somebody else breaks your code.

Now, within Google, we've done a lot of things to defend against this. There's a practice within Google of doing a lot of defensive testing, for example. So you can't commit unless it passed all the tests. So you write test to make sure that nobody breaks you.

On the other hand, there's a lot of people who really still don't like that, just the chance that something could ever break them. They want to have the control of knowing what version they're on and updating it at the point when they choose. So we saw people who really felt that having those stable dependencies was hugely beneficial.

There was even a really interesting minority viewpoint that the ability to change your library at will and update all your clients at the same time, like we do at Google, is actually detrimental to API design. This is not my particular viewpoint. I always have a trouble like describing this one, because I so strongly believe against it. But the viewpoint is that by knowing that as soon as you write out a version of your library that someone's going to use it and there's always going to be somebody using version 1.0 even when you're on version 6 means that you have to think carefully about your API design from the beginning and you can make mistakes.

[00:25:05] JM: I mean, that granted, but we know how to do API back compatibility. Don't we? Isn't that a reasonable constraint to impose upon developers?

[00:25:15] CJ: I think it is. I personally view it as a library deciding myself [inaudible 00:25:19] in that role, I view it as, "First of all, I'm a software engineer and I'm a human and I'm an idiot. I am going to make mistakes in my API design." There's going to be things I don't anticipate. Additionally, the environment is just going to change. That software engineering, the context is always changing underneath us. So we need to be able to adapt their libraries and our APIs accordingly.

Now the cool thing about Google is we don't always have to be exactly backwards compatible, because you can update all your clients as well. Because, remember, we do have access to the entire codebase. There's actually a philosophy within Google. It's a weird philosophy here, that when you are a library owner, the onus is on you to update your dependencies, the people who depend on you.

[00:26:03] JM: But it's not strictly like you need to keep your dependencies API back compatible. You could conceivably update your service as well as all of the clients that depend on your service.

[00:26:14] CJ: Yes, and I've done before. I mean, I've made a mistake in a library before that I was deciding. It was all these mistakes that I should've known better, and even my manager at the time questioned me on it. It was like, "You really want to do that?" I'm like, "Oh! It's going to be fun."

Three months later, I was really regretting that decision. At that point, I had 900 clients. So I had to go in, change to create a new version, and then I updated all of my clients to use the new version.

[00:26:41] JM: Were you able to do that programmatically?

[00:26:44] CJ: Oh, yeah. We have lots of tooling around this. This is where things like our code indexing tool again come in really handily, because I could easily find all of my clients, search

for them, automatically update their code. We have tooling that will automatically send out all these updates to everybody and make sure they're property code reviewed, that they go through all the tests. If there's any test that fails, that will notify me so I can go and fix it myself. It's an amazing piece of infrastructure.

[00:27:12] JM: Now I see why people miss working with the tools within Google. That's kind of hilarious. I really have to side with you on this one, because I have worked – I think I've literally worked at three places that all had multiple versions of Java running at the organization, and I have not worked at many software companies. It's like you're in this constant stroke, "Okay, are we updating from Java 6 to Java 7? Oh! No. No. No. No, sorry. We're talking about the part of the organization that's updating from Java 7 to Java 8," and it's just a nightmare, and that's just Java.

[00:27:50] CJ: Yeah. The amazing thing here was I got to show up one afternoon and there was an email in my inbox saying, "Congratulations! You're all now on Java 8. You may now use Lambdas at will." There was even somebody who'd even written tooling to help you automatically update your code to use Lambdas in obvious places, and it was so awesome.

[00:28:11] JM: Amazing. You're revealing a world of tooling, interesting tooling opportunities here for me. Is there anything else that like stands out as kind of a useful or interesting tool that you have seen anecdotally within Google, like help with this kind of at scale repository management?

[00:28:30] CJ: I mean, I think the big things are all around our code navigation features. I mentioned this code browsing idea, the ability to just go find a piece of code, quickly search for it and then say, "I want to see everyone who's using this code so I can find examples." Maybe because I am trying to use this library myself or I'm trying to understand like an edge case of how I'm using this library when I see other people who are doing something similar. It's seconds, a couple of clicks to do that here. That was actually the big thing I missed when I was – I had spent a couple, about a year, working on open source project while I was at Google, and I swear I felt like my productivity dropped I half, because I couldn't easily navigate my own code, much less the code of the libraries that I'm depending on.

[00:29:19] JM: How did you wind up in a situation where you're doing these like studies of developer tooling? Take me through the kind of winding path that I'm sure why did you do that kind of situation.

[00:29:32] CJ: Yeah. So, I actually have – So my graduate work is in software engineering. So I studied developer tools and studied primarily software frameworks and static analysis tools in grad school. When I came to Google, I was put into the developer tools group and I was actually on the static analysis team originally.

So we actually had a paper on that too. It's called Triquarter, and it's our really fun static analysis tool that runs across all of our code on every single commit and provides engineers with useful warnings about their code.

From there though, once I was in the developer tools group, there's actually a few of us in that team who had prior research experience in software engineering, and we were noticing that our group as a whole, the developer tools organization as a whole, was looking for kind of what's the next big thing we can work on. What's the next step we take to improve engineering productivity?

Up until this point, it had been kind of random projects that people thought of, and you were getting the low hanging fruit. But we were hitting a point where it's not clear what we do next. Our VP actually start asking these questions of, "Well, if I have 10 people to put on a project, should I put those people on improving our build system or should I put them on the code review tool?" So our team was formed around this idea of help our executives understand where they should best put their resources to improve productivity.

[00:31:06] JM: Hilarious. Static analysis is one of these areas where I had no idea this existed before I started a podcast about software engineering, and yet it has turned out to be just this tremendous field. I think the most recent show we did about static analysis was with an interview with a company called FOSSA, that basically does static analysis around whether you might have licensing issues in your open source repositories.

[00:31:35] CJ: Oh, that's a nice one.

[00:31:36] JM: Yeah. What are the big unsolved problems in static analysis?

[00:31:41] CJ: That's a good question. So I could go for the more theoretical answers here or the more practical answers.

[00:31:47] JM: Let's do both.

[00:31:49] CJ: So the theoretical things that people are still working on are in an area called alias analysis. This is how do you know which objects are pointing – Which pointers are actually pointing to the same object. It turns out that this is a kind of foundational piece to really make any static analysis system work. It's really tricky to make this work and make it work at scale so that you can actually analyze a large codebase as supposed to just a little toy problem. This is something that people are still working on.

Once you're able to solve that better, it will actually enable a huge amount more static analysis tools. Practically speaking, there're a lot of problems that I think that our team here at Google solved a lot, but there are still a lot of problems with getting the right level of utility out of static analysis tools. We see that there's a lot of cool tools, for example, in academia. But when you look at them in practice, people don't use them, because either they have too high of a false positive rate or they don't actually have a high false positive rate, but people don't understand what to do about the problem. So they see an error message and they go, "Okay. What is this even mean?" If it takes you 15 minutes to figure out what it means and it turns out that it was a false positive, no one's going to use your tool.

One of the things we did at Google to try to make static analysis more useful here is we actually started focusing on the simplest possible errors that had the simplest error messages that we thought people actually want to respond to and fix. We were aiming for less than 10% false positive rate. I think we're down to 3% false positives now.

So we've built up over the course of five years now a lot of trust in the static analysis tools, and there's now hundreds of analysis tools running within Google. But we take a view of trying to make them very practical. There's actually – It's kind of funny that I spent a lot of time in

academia working on some pretty complex analysis tools. I did a lot of things with data flow analyses and to procedural analyses, and it turns out that in most cases the simple dumb analyses work out really well in practice.

[00:34:04] JM: So the alias counting thing, why would you need to do that in a static analysis tool? Because like don't garbage collectors just do that at runtime and it works fine?

[00:34:16] CJ: But that's at runtime. So at runtime, sure, now you have precise perfect information. That's the benefit. So a garbage collector would be something we'd call a dynamic analysis tool, because at runtime you have precise information. But you only have precise information over the path that you are running on. If you view all the paths through your code, you've got the one path you're going through at runtime.

Static analysis has a lot of interesting tradeoffs where you're looking now at all the possible paths, including even some that might never occur at runtime, because no user ever inputs that kind of data. So you're trying to handle all those paths. Additionally, you don't have precise information help. You'll lose information about which objects point – Which pointers points to which objects and whether they're the same or not, and that starts to cause problems.

Now there is – I should mention this too. There is some really interesting work I think that uses both static and dynamic analyses together, and I think we're going to see a lot more from that around from academia and hopefully eventually in industry tools as well where you do things that you use a dynamic analysis to understand what are the possible paths that people take through this code. What are the possible inputs? Then you use that to feed into a static analysis to help make it more precise.

[00:35:30] JM: Wow! Then you get to like machine learning-based applications kind of.

[00:35:35] CJ: Kind of. I guess I wouldn't.

[00:35:36] JM: Because, I mean, there's probably so many dynamic paths you could potentially find that you take the loss function or you just basically feed the data from the dynamic analysis back into the static analysis tools. So you get like update.

[00:35:53] CJ: I've seen this go the other way too, where you can use dynamic analysis to feed static analysis. Actually, one of my co-authors and I did that in grad school, where I had a system that was a static analysis that would check your – That you're using a library correctly, client-side, that you're using a library quickly. But you have to create specifications for that. Creating specifications is hard. It's a lot of work. Nobody wants to do that, but he had a dynamic analysis tool that would automatically generate the specifications. So we could make that work out so we could feed one into the other and then you could go back again.

[00:36:26] JM: This is kind of like what a protocol buffer is or like a GRPC thing, or is that – Am I misunderstanding?

[00:36:33] CJ: So, a little bit different. A call buffer would be just the data storage. What I was meaning by this is we're looking at the API for how people use maybe a protocol buffer.

[00:36:46] JM: Okay. What was it about static analysis that was so intriguing that you put your graduate efforts into that?

[00:36:54] CJ: So that's actually a really interesting story. I was in underground. I was working as a software developer at a couple of companies throughout my undergrad career, and I remember having this day where I was working in, I believe – Yes, I was writing some C# code and I get a bug report from our tester, and it's like a no pointer exception.

I go and open up the code and I'm looking at it and it was so damned obvious. I had done just like a really stupid thing. I had set something to null and then immediately, like immediately try to use it. I'm just going – And I kind of was a little annoyed with myself, because this is a dumb thing I should've caught, and I also should have of course had better test. But then I thought to myself, "How come the compiler didn't catch this? This is so blindingly obvious?" So that's what kind of caused this interest in static analysis tools, is that I felt like the compiler should be doing more for me.

[00:37:56] JM: In that specific case, like good point. Why didn't the compiler catch that? Is there something that is fundamentally so hard about looking at two lines of code and being able to identify that a no pointer exception is going to occur there?

[00:38:12] CJ: So that was a really great example of one where that could have been done easily, and I have not written in C# in a long time. I am certain that my colleagues at Microsoft have actually already fixed this. But there are cases where it would be difficult, and this is where that alias analysis comes back into play, like, “What if in between those two lines I had made another call and that did get initialized somehow?” So it’s important to know where pointers are actually at.

[SPONSOR MESSAGE]

[00:38:50] JM: DigitalOcean is a reliable, easy to use cloud provider. I’ve used DigitalOcean for years whenever I want to get an application off the ground quickly, and I’ve always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That’s a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his

interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:40:57] JM: Not to go further down the static analysis path, but I find it kind of interesting. We did a show kind of recently around this company that was like a security company that did static analysis on Ruby. I think was basically on like at the latest possible time. So I think Ruby code like turns into C code, for example, or I think it compiles down to C or compiles down to C struct at least.

In any case, they did static analysis on the lower level interpretation that the code compiled down into rather than the higher level. Have you seen any like interesting – I don't know, studies or anecdotes in the world of like doing static analysis on the high-level language versus like the byte code or something?

[00:41:44] CJ: Yeah. I mean, there's plenty of tools that do each of those. There're pros and cons as far as building a tool that does that. One of the benefits of working for like if you were on Java working on the byte code level, is there's fewer cases to handle all of the syntactic sugar gets compiled down. It's much more straightforward to actually write an analysis tool. There're also benefits if you can compile multiple languages into the same format. So I know there're companies out there that are doing that and they idea is that where they can trace code paths, go between multiple languages, or you can write a general checker that works for many languages. So there can be benefits there.

The downside of working at kind of more of a byte code level is it makes it more difficult to write error message that the engineer can actually understand. Yeah. At least in Google we typically are working on source code level.

[00:42:42] JM: Yeah. Coming back to the research area, are there other studies you've worked on, or is the only study you focused on this repo question?

[00:42:53] CJ: No. We've actually got a bunch of other studies, some of which have been published already and few that we've got in the works. So one of the ones, for example, that we've published recently is looking at productivity factors. So, what are the things that influenced someone being productive?

We ended up – One of my colleagues went into the literature review going back to last 50 years of research on productivity, and he pulled out a whole bunch of different things that the research literature had said it makes people productive or not productive. We then filtered through that, combined things that were the same and ran a survey at several companies actually. So it wasn't just Google.

Also, at Google, we ran it also on not only software engineers, but we also looked at quantitative analysts, because they are another form of knowledge worker and we want to see what the differences were, software engineers and other knowledge workers. It was a really interesting study. We found that there were a lot of things that influenced people's – This is all self-reported productivity by the way, I should say. There's like no objective measures here, because it's really hard to objectively measure productivity. But there were a lot of things that correlated with people's self-reported productivity, and they were in fact different at each of the companies and they were even different even at Google between the software engineers and the quantitative analysts.

[00:44:17] JM: So result inconclusive?

[00:44:20] CJ: Well, I don't know if it's result inconclusive. It was useful to see that there was those differences though, because that might tell us something about how we focus on each of these user communities. It means that maybe we can't just say, "If you do the following five things, that's going to make your engineers more productive. Maybe your engineers have different needs, and so you have to rerun this study for them."

[00:44:42] JM: Have you seen anything conclusive around this open office plan, versus closed offices, versus remote work? Kind of like high-level circumstances mapping to productivity?

[00:44:42] CJ: I have not been keeping up with that research. I know it's very popular amongst like the Hacker News crowd right now. I've actually not been keeping up as much with that research. I probably should get into it and find out what's going on there though. I've just been seeing the headlines though.

[00:45:10] JM: Yeah. Yeah, that one's tough. I mean, that one is just really hard to measure. It's kind of like the same thing with the question of like if Google would've gone in the direction of microlyths, arbitrarily, would they have ended up with just great tooling for microlyths? There are these companies that say like, "Hey, we're remote first." GitLab for example, and like, "We just do everything over Slack channels and things are fantastic. Why don't you all join the remote club? It's just very hard to know like, "Okay. You're working on a different kind of software. Maybe there's something about the culture or the culture of the founders that leads to a remote culture being holistically viable." I think I guess my meta point here is it's very hard to do this kind of research and have real confidence in your results.

[00:45:59] CJ: It is. There's a lot of variables from one company. In instance, you mentioned the context, whether or not – How often, for example, people need to collaborate for this type of project.

[00:46:10] JM: Any guiding principle there when you're thinking about like necessary sample size or – I don't know, other kind of parameters that will allow for a study to be even just in the foundational level, like a viable sample set or a viable study architecture?

[00:46:28] CJ: I think it depends on the type of study you're going for. There's I think sometimes a view that – And especially within software engineering research, that we need to always have the best possible studies with a control group at a treatment group, and we've narrowed down our independent variables and also have a large and on top of it. That's just not realistic. That something you can do in medicine. But even in medicine, they start with other types of studies. They start with small case studies, for example.

If you look at medical research, especially in the cases where it's a very rare disease, they're not doing large end studies. They're small case studies and they buildup a kind of foundation of case study work before they can start making more interesting hypotheses and saying, "Okay.

This seems to work across all these groups, or it doesn't. It only works in certain groups. Now let's make some new hypotheses and start having more experiments." I think that's true for software engineer research as well.

Because I'm at Google, I'm currently in the case of I'm doing kind of effectively a large case study within Google on all of our research. On the other hand, my colleagues in academia are frequently doing large case studies on open source. Every once in a while [inaudible 00:47:41] able to do a study across multiple companies, and we kind of end up pulling all of that together, and across several different research papers we're able to start seeing things emerge.

[00:47:53] JM: So the thing you alluded to earlier, where you would like to let the executives have more insight, statistical insight, anecdotal insight, white paper style insight, into are they allocating engineering resources properly. Was there any kind of reallocation or insights or, I guess, action insights that came out of the repo study?

[00:48:20] CJ: I think the repo study, I don't think we did have any other than just like everyone's happy with a status quo. So let's keep it going.

[00:48:27] JM: Yeah, I guess that's true.

[00:48:29] CJ: I mean, it was so blindingly obvious that we should just kind of keep this going, that we just left it like that. I have had other studies though where we've either at the executive level been able to give some information and help them make a decision and maybe change course. Honestly, a surprising number of cases, our recommendation was backed with, "Keep the status quo. Everyone is happy with this. It's like this for a reason."

But even had on a more microlevel, we've had times where we've told the whole team of like a tools team, "Hey, you should work on this problem rather than this other problem." So like some of the programming languages teams come to us regularly and ask, "Which of these following three problems should we be prioritizing this year?" We're able to use our survey data and our log so it would actually give them some useful insights on where they could be most impactful.

[00:49:16] JM: There was an interview I did with somebody from Facebook recently, and Facebook actually has also – I think, to my understanding, has also thrived with a mono repo.

[00:49:28] CJ: Yeah, they have.

[00:49:29] JM: And the way that they've done it, my understanding is actually with probably significantly less tooling, or at least I would guess. I've read the like feature flags play a really important role. Like you kind of like manage the different areas of the code base with feature flags, which is kind of like a more late binding – I don't know. I shouldn't be speculating on this.

But anyway, I guess the question I want to ask is like, "Do you think the broader industry can learn anything from this study, or do you think it's too Google specific, or gigantic company specific to say if people should go mono repo or not?

[00:50:08] CJ: I think there is something to be learned here. I would not, for example, if I was in another company, I would not take what we've done and say, "Oh, Google is using a mono repo. Therefore we should too." Clearly, I think our number was something like 88% of Googlers say they prefer their mono repo. I wouldn't take that number and say, "Therefore, every company should use a mono repo."

But I would take is looking at the benefits that the Googlers have identified to a mono repo and the benefit they've identified as the multi repo and these tradeoffs and attentions we identified. That, I think, does apply across other companies. So I think that's where you can kind of make the engineering choice of, "Let's see. Which of these tradeoffs apply best for my company?"

[00:50:51] JM: I don't know if you'd answer this, but there must be some repos within Google that have to have to be permission. I mean, it's almost like a zero-trust networking kind of thing. Is there a permissions model for this mono repo?

[00:51:06] CJ: The permissions model is primarily around who commits to the code and who approves those commits. So I can commit to any part of the code base or I can at least attempts to. But the owner of the code is the one who says, "Yes, I'm going to accept this commit."

[00:51:21] JM: You can see all of it.

[00:51:23] CJ: And you can see all of that. Then we do have some cases where people have separate repos. So not quite all the code base is in one mono repo. One of the classic examples actually is a lot of our open source projects. So Android and Chrome are both in their own repos, although they're fairly big projects in and of themselves. Then we do have several GitHub projects for a lot of for open source projects.

[00:51:48] JM: Oh, good Lord! I'm thinking about the Android and Chrome repo management. That sounds like a can of worms or they each sound like cans of worms respectively. Is there anything notable about – How did all those different versions of Android get managed?

[00:52:06] CJ: I'm actually not working with the Android team right now. So I'm not entirely certain. But I do know they have a git-based repository.

[00:52:14] JM: Okay. Yeah, fair enough. Anything notable about the open source, the interaction between the open source codebases and the main Google code bases. I don't know, integrating those two kinds of somewhat disjoint domains?

[00:52:29] CJ: Yeah. There's some tooling around trying to keep things in sync. So if your project, for example, internally depends on open source project. There's a work around how to keep this projects in sync together. But it's pretty straightforward tooling actually that we have there. This is kind of our props. Our solution for everything at Google; just add more tools.

[00:52:48] JM: So we're seeing – I think when a company reaches a certain size, I think it often behooves the company to set up some sort of research thing like what Google has done. Like you have Google research, your Facebook research. I think companies like Stripe probably have something nascent around that looks like research. Microsoft research obviously. Do you have any general principles for – Let's say I'm a company and I'm just reaching like the thousand person mark and I've got really good traction, and like everything is going really well. I'm not constrained by capital. I can hire a research team. Any advice for setting up a research team or managing an in-company research team?

[00:53:32] CJ: I think the big thing with us that made us successful was we were very careful about how we grew this team and the people we added to it over time. One of things we did is we did start small with people who had prior experience with software engineering research. But over time we've added to group and we've been especially adding people who have social sciences background.

So our team right now, about half of us are software engineers. Of that group it's about software engineering research background and half the people have more of a mathematics background. Then the other half of our team actually is more social scientists. We've got people who came from cognitive psychology, neuroscience, behavioral economics, and they bring a lot of rich research methods that are unique, and they're thinking about problems in a different way. Once you've got a behavioral economics on your team, they think about, "How do I tweak things to make people do the right thing without them really realizing it? How do I set up the incentives?"

You get someone who's a cognitive psychology and they're noticing the differences between how people think about their colleagues and how that might affect their work product. So I think that has been very beneficial to us. I know some other companies also try to get people like that as well.

[00:54:52] JM: Kind of envious of your lunchtime conversations.

[00:54:54] CJ: They are. There're some very, very, very interesting lunchtime conversations. I've learned so much on this team.

[00:55:00] JM: So last question; what kind of studies are you curious about or like what do you see is the future of your research to the extent that you can talk about it?

[00:55:10] CJ: Well, we've got some studies hopefully coming out in the next couple of years. We're going to be working on things like looking at build speeds and how that affects productivity. Yeah, I know. It's a hot topic.

[00:55:21] JM: My code is compiling.

[00:55:23] CJ: I know. We're also – We've got some fund research I think the code readability process at Google that we're going to try and push out the door, because I think there'll be some really interesting lessons there for people outside of Google. More broadly, I am really interested in getting more involved in trying to understand things like how context switches affect engineers and how often do we context switch, and how difficult is it to get back on context?

There're actually a couple of papers that have been coming out in the academic community about this where they've been trying to study GitHub and looking at like, "Within GitHub, how many projects does an engineer work on a given day or a given week, and are they switching back and forth between them quickly or are they working on one for two days and another one for three days? How does that affect productivity?

[00:56:10] JM: This is actually what made me need to leave my career as a software engineer, a corporate software at least, is because Cal Newport has this thing he calls Deep Work. Everybody needs to be doing deep work. They need do have some segment of their day where they're not context switching. They needed to be completely focused on what they're doing. Yeah, I happen to really enjoy shallow work with tons of context switching, and that's kind of a rare work environment to find in my experience. But I'll be very intrigued to see what you find in your context switching analysis.

[00:56:43] CJ: Yeah. I'm going to be curious too. One thing we do have some evidence of is that there are differences from one person to the next here. So there are people like you who would rather be switching between a lot of different projects and doing a more shallow work and there are people who were like, "No. No. No. I need to work on a single project at a time. No interruptions until this thing is done."

So it'll be interesting to see how that plays out with maybe how we change up again, how [inaudible 00:57:08] with the tooling. How do you maybe even go into open office spaces? How do we change up how people work based upon their preferences and what makes them personal and the most productive? I don't think there's a single right answer here though.

[00:57:21] JM: Well, I completely agree with you there. Ciera, thank you so much for coming on the shore. I've been really enjoying the conversation and I look forward to seeing your future research.

[00:57:29] CJ: Thanks so much.

[END OF INTERVIEW]

[00:57:35] JM: GoCD a continuous delivery tool created by ThoughtWorks. It's open source. It's free to use, and GoCD has all the features that you need for continuous delivery. You can model your deployment pipelines without installing any plugins. You can use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your cloud native project. With GoCD on Kubernetes, you define your build workflow. You let GoCD provision and scale your infrastructure on-the-fly, and GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn how you can get started.

GoCD was built with the learnings of the ThoughtWorks engineering team, and they have talked in such detail about building the product in previous episodes of Software Engineering Daily. ThoughtWorks was very early to the continuous delivery trend and they know about continues delivery as much as almost anybody in the industry.

It's great to always see continued progress on GoCD with new features like Kubernetes integrations so you know that you're investing in a continuous delivery tool that is built for the long-term. You can check it out yourself at gocc.org/sedaily.

[END]