EPISODE 646

[INTRODUCTION]

[0:00:00.3] JM: Chad Fowler was the CTO of Wunderlist prior to its acquisition by Microsoft. Since the Acquisition of Wunderlist, Chad has become the general manager of developer advocacy at Microsoft. He also works as a venture capitalist at BlueYard Capital, an early stage investment firm.

I had a lot of fun talking to Chad because he can move seamlessly between talking about disparate subjects, like cloud computing, investing, cryptocurrencies, and music composition and he has novel things to say about all of these. When Chad joined Wunderlist, he helped start a large refactoring of the software architecture, which makes for a great story. Then he helped the company navigate to a successful Microsoft acquisition.

We started off the conversation with the story of that re-architecture of Wunderlist and then we talked about the current opportunities that he sees in front of Microsoft. Chad gives his perspective on Kubernetes, functions as a service and how developer tooling might evolve in the near future. After talking about the near term of developer tooling, we talked about the distant future; bug bounty market places on the blockchain using GitHub repositories to train machine learning models about how to write code, the comparison between music collaboration and software collaboration. This was a wide array of topics, but Chad was equipped to discuss all of them, and that's because he works at Microsoft and he makes large investments in the future through venture capital, and he studied music when he was in school. So it's really a lot of random stuff in this episode, but was really entertaining from my point of view and I think you'll enjoy it as well.

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

Thank you.

[SPONSOR MESSAGE]

[0:02:23.7] JM: We are all looking for a dream job, and thanks to the internet it's gotten easier to get match up with an ideal job. Vettery is an online hiring marketplace that connects highly qualified job seekers with inspiring companies. Once you have been vetted and accepted to Vettery, companies reach out directly to you because they know you are a high quality candidate. The Vettery matching algorithm shows off your profile to hiring managers looking for someone with your skills, your experience and your preferences, and because you've been vetted and you're a highly qualified candidate, you should be able to find something that suits your preferences.

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

Get started on your new career path today. Get connected to a network of 4,000 companies and get vetted and accepted to Vettery by going to vettery.com/sedaily.

Thank you to Vettery for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

[0:04:07.4] JM: Chad Fowler, you are general manager of developer advocacy at Microsoft. You were the CTO at Wunderlist and you do several other things. Welcome to Software Engineering Daily.

[0:04:18.2] CF: Thank you very much, I'm excited. I'm a big fan of the show.

[0:04:20.6] JM: Well, thanks for listening. Actually, you reached out a while ago and we met up for coffee and had a really engaging conversation and I'm looking forward to continuing it, because it was one of those conversations where you sit down and kind of the topics overflow the amount of time you've allocated for the conversation, which is always a good sign.

[0:04:40.1] CF: Agreed, yeah. I felt like we probably could have gone the rest of the day and wouldn't have known what we talked about by the end.

[0:04:46.7] JM: Right. So you are CTO of Wunderlist, and Wunderlist is a to-do app, a productivity app that ended up getting acquired by Microsoft. I was a user of Wunderlist for a pretty long time. I really enjoyed it. So you joined Microsoft and now you're working on the Azure side of things. You're also doing investing, you write to music, and I'm excited to touch on all of these different areas. But let's start with Wunderlist, because I think there are some good architectural lessons from that experience that I've heard you discuss before. When you joined Wunderlist, what was the application architecture like?

[0:05:24.8] CF: It was the usual two box architecture, where you have a client in one box and a server on the other. It was very monolithic. The clients had just been rewritten in native code and they were just fabulous, but the backend was Rails sitting on top of PostgreS, I think at the time. Maybe MySQL, but sitting on top of a monolithic database. A bunch of pretty strange design decisions in the Rail service itself. It was just an API server. It wasn't like a web app, but some really, really interesting stuff with like in-memory – What are they called at the time? Basically, like Rails plugins, middleware's. That's what they're called. So that was the composition strategy. That and mixins all over the place.

[0:06:14.4] JM: I think your experience involved some re-architecting of that monolith, but from a prescriptive or proscriptive point of view, do you think there's anything inherently wrong with a monolith? Because some of the conversations I've had with people recently have kind of gone back-and-forth around that. You can over index on micro services, right?

[0:06:38.0] CF: Oh, absolutely. Yeah. I mean, I think once we had a name for micro services, thanks to Fred George, that's when things started going wrong. It's not his fault, because you

kind of need that to spread an idea. But as soon as you have a name, just like Agile, BDD, dev ops, they all basically mean nothing at the end and become an anchor for bad behavior.

So micro services, you can over index, you can do all kinds of stupid, crazy stuff just in the name of micro. Is there anything inherently wrong with monoliths? I'm still going to say yes. I think there's probably something inherently wrong with pretty much any architecture though. So maybe that's a non-answer, but there are some things that make monoliths hard to deal with at least given modern technology.

For example, being able to isolate performance of different pieces of a system and actually understand which thing is causing problems. IT's really hard in a monolith no matter how well architected it is. Whereas in micro services, that part is easier. So I would say the coupling aspect of monoliths is inherently negative. Maybe not inherently wrong, but inherently negative.

[0:07:51.5] JM: Well, and that difficulty of finding out what's going on. So I saw a talk that you gave when you were talking about what the state of things was at Wunderlist when you joined. It sounded like that you basically did chaos engineering to make things understandable. You joined the company and you just started making things fail and fall over in order to understand the code paths that cause those failures. Am I understanding correctly what happened?

[0:08:22.3] CF: Sort of, yeah. Yeah. So I took the job three months before I was actually able to move, and I had to move to Germany to do it. So in the meantime, between accepting the job and actually showing up, they had released a total rewrite of their – Actually, everything, but including the backend. By the time I got there, the team was just shell-shocked. It was like PTSD. Everyone was afraid to touch the system, because every time they deployed or did anything, it would crash basically. They have had many successive days of downtime at certain points in a two-month period.

So I got there, it was kind of a weird first day sort of experience where I was shown a room where I would sit, and that was pretty much it. I guess it's like, "Hey, you're the grown up. You go figure it out." So I had a computer maybe in a room. I went into the room and started talking to people and realized they were scared. So I got access to the AWS console. We were using

AWS at the time and started removing servers from clusters until we hit the point where the system just started to explode. It didn't fully explode, but started to.

Of course, everyone's kind of pissed because they are thinking, “Why have you just broken the system? Wonderful that we just hired you, great.” But what we did then it is we brought it back up together, and it wasn't a disaster. It was just scary. We started doing that often, because what would happen is before I got there, and I think they finally propped it up to the point of working and they were afraid to do anything. So they didn't really understand why it was working then or why it didn't work when it didn't work. They got it working. They didn’t want to touch it again.

So I got us used to recovering. It was good for me, of course, as it was for them in terms of learning my way around. But I think getting used to recovering and being comfortable in that space and being able to think clearly and develop the toolset for inspecting the system, and then – Like I was talking about with monoliths, starting to develop an understanding of the failings of the architecture, because in some cases there are no tools that help you recover as you just can't figure out what's happening in this monolithic rails app. That was all part of the process and that's why I did what I did.

[0:10:39.5] JM: And what were the responses that you made to that so that you learn some problems? What happened with the re-architecture? Did you just start breaking out into services or was there anything else you tried to bring into it?

[0:10:55.2] CF: It was both an incremental and a radical approach. I decided pretty early on for a variety of reasons, some of which were about how the system was architected and how the public interfaces were defined. There actually wasn't fully incremental way to fix the problem, and it would take way too long to explain. It would be boring probably to explain why that was. But they had created a system that was not incrementally replaceable.

So we started by doing some simple things like looking for hotspots in the code, in the database, and breaking the monolithic database into what did actually end up being the ultimate design, which was a stupid sharding where we just took all the tables of the whole system and put them in separate databases so they could at least have different I/O constraints and we could scale them all separately.

That sort of thing, and then in parallel, we started a complete re-architecture and rewrite, which we launched about a year, a little more than a year later after the scene that I was just describing of bringing the system down over and over again.

[0:11:59.0] JM: So the re-architecture and rewrite, do that consume the resources of the entire engineering team or were you also releasing new features at the same time?

[0:12:05.9] CF: For a while we were releasing new features. I guess several months went by where we were taking what was a disastrously unhealthy system and making it passably healthy. At the same time, mostly the client teams were working on new features, like we introduce Wunderlist Pro, which allowed us to do billing and have pro features. We introduced file attachments and comments, and it was the comment system, that was our first real step at creating a new service that looked like the full rewrite would eventually look. So we were able to do something incremental and we added a new service, and that's where we played around with and prototyped what would ultimately be our real-time sync protocol as well.

But then at a certain point, we just had to get dark for a long time, and it was scary. I mean, I wrote a blog series that got linked all over the place in 2006, basically declaring I was done with rewrites and I've done too many of them in my career, and here's all the reasons you shouldn't do it. Of course, I spent the next 10 or so years continuing to do them. This was a huge one.

Yeah, when we finally released it, it was a big rewrite, but we had figured out a way to piecemeal at least API calls. So we created a whole new architecture for APIs, and then in the old monoliths we replaced some of our rails models, which are just DOM Ruby classes that are usually like ORM classes sitting on tables. We replaced those with classes that in Ruby we're doing rest calls to our new APIs so that we were at least in production, exercising the new architecture in some way before the full launch. So it ended up being this weird piecemeal of a full rewrite and incremental over the course of a year.

[0:13:56.6] JM: So people loved Wunderlist. Wunderlist, it's like one of those things that's like Trello, or Dropbox, or Slack. Kind of one of these things that sits somewhere between the consumer and the enterprise and plays a little bit in both of those environments. So it eventually

got acquired by Microsoft. That was in June 2015. What was the process leading up to that acquisition?

[0:14:24.0] CF: I think it took six months total, and it starts –

[0:14:28.9] JM: The whole conversation and everything?

[0:14:30.9] CF: Yeah, exactly. Usually what happens is a when a big company like Microsoft, and this is true for most big companies I think, is interested in talking about acquisition with a startup. They will first reach out and say something like, “We want to have conversations about partnership,” and it will feel kind of vague and it's usually someone who has a title that looks like biz dev or corp dev, and usually the biz versus corp thing is the telling factor. Corp dev typically means acquiring companies. So we got contacted like that.

We were also in the mode at the time where as a management team we had decided we need to think about the future. We had raised a series B from Sequoia in 2014, I think, or 2013, and it was about time to figure out whether we try to buckle down and get a profitable or raise another round or resell the company.

So some of us on the management team had decided selling the company was probably the best choice and we’re pretty much already there, and I was one of those people. So we were already excited to talk to Microsoft, because we’ve been watching the acquisitions they've done, like Acompli and Sunrise, and just the changing culture, change in design aesthetic, user-friendliness, all those sorts of things that everyone talks about in the Satya Nadella Microsoft world. So we're really excited to talk to them when they came along.

The way that process works is you have a few conversations that are really vague. Eventually they say, “Yeah, we’re actually talking about acquisition here. There is a deal sponsor in the company, which is the person in a business unit that really brought this up and said, “I want to acquire this company,” or “ I want to acquire a company like this.”

Then you go through this intense dating experience basically, and it really is like that, because no one is really allowed to say, “I like you. Let's do this,” in the same way that when you maybe

are shy and/or first getting to know someone. You're not really direct and blunt, like, “Okay. I think we should be together.”

So we went through several weeks there, or maybe even a couple of months, so sort of going back and forth. We made some visits back and forth, and then you declare your actual intent, Microsoft in that case. You sign a contract with a letter of intent and then you could start going through the process intensely of due diligence where you've already agreed on a purchase price and it’s really just making sure that the company is getting what they expect and there aren't any problems under the covers that they haven't seen.

[0:17:16.3] JM: Is it hard as a negotiation, or is it pretty – Can you sort of do the evaluation and people come to an agreement or – How difficult is that process? Does it feel like it's just kind of a back-and-forth and asymptoting towards something that's sort of win-win? Does it feel like somebody's trying to get the better of somebody? Because me it seems like the successful acquisitions are going to be the ones where the transaction happens and nobody has remorse.

[0:17:44.9] CF: I don’t know. It depends on the acquisition. It depends on the acquirer and the company they’re buying. I would doubt there’s ever been there – Or I would say there has rarely been a really easy one where you just say, “Hey, we want to buy you for this much,” and the acquired company says, “That sounds good,” and you go with it. There’s always some back and forth.

Most likely, what I know from our experience, and I've been involved in companies that have been acquired twice in the management teams. Usually there is a range that everyone expects on both sides, and if you're anywhere inside that range, then it's probably something like buying a house. You’re going to triangulate on a number, and everyone can kind of guess what it is. If expectations are fully off, it’s probably not going to be the case that the transaction goes through, honestly.

[0:18:42.5] JM: So what was the integration? After the acquisition happens, you figure out, “Okay, here's our synergies, here’s where we’re going to integrate. Here's where we’re going to kind of stay as separate product? What kinds of re-factoring had to be done? What was life like during that integration period?

[0:18:59.6] CF: You might even say it's still going. I don't know. In the beginning, the guy who led the deal and who became my boss, his name was Eran Megiddo. I guess it still is Eran Megiddo. But he is a vice president that has to do app now, is what it's called, OneNote education, and the whiteboard software that Microsoft has just released. He kept saying as we are leading up to the acquisition, "I don't want to get in your way. You just keep doing what you're doing. You're doing a great job. Don't let us slow you down."

That sounded great, obviously. When you're being acquired as a separate company you think, "Yeah, I want some freedom." When we got in, it turned out that he was being absolutely honest, even to a point where I almost wished that he wasn't like that.

So we had to sort of find our way around the company. He helped, of course, but he really wasn't dogmatic at all or draconian about what we needed to be doing. It became really like a business development exercise, and I mean that very literally. There was a guy on the team that ended up playing the role of business development inside the company. I took over as general manager of the team instead of being a CTO anymore pretty much right when acquisition started. So I led the whole thing, and he reported to me. It was really like we're just going around to different teams in Microsoft talking about the things that we wished we were able to do when we were separate and small, and we thought maybe now we can do.

As we talked, we had built a story that became cohesive and literally created a pitch deck that we went around with and started showing to everyone and got people excited. So our vision ended up being – I don't know if it really surfaces in exactly this way, but our vision early on was that tasks become the connective fabric between productivity applications. So they connect the suite of product of the applications together, because they're the things you need to do. That was a pretty easy thing to sell. Microsoft has the Microsoft Graph API. It really is like you have this graph of work objects already if you're using Office 365.

No one told us, "Okay. Now switch your –" and I didn't talk about the architecture that we landed on, but we had a crazily heterogeneous micro service architecture with servers dying and being brought back to life constantly. This sort of cellular regeneration metaphor, and we are running on Linux. Everything was Linux. Haskell, Elixir, Ruby, all sort of languages behind it. No one

said, “All right. Now consolidate this and run it on .net on ASP,” or something. There is nothing like that.

It was more the company trying to figure out how best to serve the architecture that we had built, and then to contribute Microsoft’s strengths to it, which would be things like deep privacy and compliance and trust, the things that are really hard to build as a small startup, but Microsoft has made an art of.

So ultimately what we decided, and it literally was like me and my team that decided it, this sounded crazy at the time. We decided to re-platform the backend, all the best stuff that we had ever built, the best part of my career, the best thing that I've ever done, throw it away and sit Wunderlist on top of Microsoft exchange as a backend. If you're like me, you think Microsoft exchange is the old email service that people were using in year 2000, and it was, but it's also a pretty amazing globally distributed key document store. It's an amazing architecture. You can find stuff on it online. Probably be a good podcast subject at some point [inaudible 0:22:59.6].

[0:22:59.9] JM: I should do that. That sounds good. Yeah, I’m going to write that down.

[0:23:02.7] CF: So we decided we were going to use that. We’re going to throw away the Wunderlist architecture, and at the same time we said, “We don't want to just rebrand. Wunderlist is not a great name for a global brand. It’s kind of a kitschy and startup-y, and maybe German doesn't work well across all markets.” So we knew we’re going to have to change the name, but we decided we're going to try and build the next version.

So like our big launch when I was there was Wunderlist III. We’re basically doing Wunderlist IV, and we’re going to call it something else and we’re going to try and create the Wunderlist killer. We’re going to take out all the stuff in Wunderlist that even though it's simple and beautiful and people love it, we think it's still in the way. It’s like not the fully realized vision. So how did we made it simpler?

So we undertook a massive project to do all that, and the output of that is Microsoft To-Do, which is in preview right now running and sitting on exchange, this compliant data store. I think it

has a beautiful simple API, great real-time sync and we're just catching up on a few features that are left. When I say we, I mean the Wunderlist team. That's where we went.

[SPONSOR MESSAGE]

[0:24:24.6] JM: Stop wasting engineering time and cycles on fixing security holes way too late in the software development lifecycle. Start with a secure foundation before coding starts. ActiveState gives your engineers a way to bake security into your language's runtime. Ensure security and compliance at runtime.

A snapshot of information about your application is sent to the active state platform. Package names, aversions and licenses and the snapshot is sent each time the application is run or a new package is loaded so that you identify security vulnerabilities and out of date packages and restrictive licenses such as the GPL or the LPGL license and you identify those things before it becomes a problem.

You can get more information at activestate.com/sedaily. You want to make sure that your application is secure and compliant, and ActiveState goes a long way at helping prevent those kinds of troublesome issues from emerging too late in the software development process. So check it out at activestate.com/sedaily if you think you might be having issues with security or compliance.

Thank you to ActiveState for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:25:54.2] JM: To-do-lists are these thing that has become fundamental to my life. Sometime after college, to-do-lists became really the backbone of how I structure my day and structure some – I mean, some bigger things in my life. When I want to get something done, it's on a to-do-list somewhere, or maybe on multiple to-do-lists. It maybe not only replicated across three different apps, but it's also on paper and it's on a whiteboard somewhere, and it's like to-do-lists – It feels like something that's very fundamental to productive people. But at the same time, the medium by which we maintain our to-do-lists seems to constantly change. Like, "Oh, maybe we

should be doing Kanban boards. Actually, no. Let's not do Kanban board. Let's not all have our personal Trellos," but maybe we keep in Slack. What are your thoughts on this? It's constantly been changing, and how do you use to-do-lists in your life?

[0:26:53.2] CF: This is, what you've just described, articulates the opportunity and kind of problem that we are trying to solve toward the end of Wunderlist and the beginnings of our time at Microsoft. I agree, to-do-lists are fundamental. Tasks themselves are fundamental. Everyone has things to do, and we all understand it. We like to interface with them in different ways. We need to be able to get to our task list everywhere, and it is a problem if you have them spread all over the place. You lose things unless you have a really great system for finding the list first and then finding the tasks.

So our goal is to try and figure out how no matter where you are, you can capture things immediately. I think we did a good job of that with Wunderlist at least as a startup, that it was trivial to put something in Wunderlist, and we had Chrome plug-ins and all sorts of things so that wherever you were, with a nice API, that wherever you were you could very easily put things in. Then to be able to recall things in a clean way and actually keep track of what you need to do.

I think the way that you project a to-do-list can be very flexible, because you can almost model anything as a list, a list of lists with items in it. Anything can be modeled that way, which means that it is therefore possible, if anything can be modeled as a list is therefore possible to project those lists in any notion of UI that is possible.

For example, one of the early experiments that we had that we just used internally with Wunderlist, but I always wanted to build before we were acquired and had a competing product, was you could think of a to-do-list as a board in a Trello type scenario and then you could move things in the board and just have that be a different visual representation. Maybe you could use statuses or tags in the to-dos to keep track of them.

I think we used tags in kind of a hacky convention-driven way. But it's still just a list. The data structure is the same, and the fact that you have the thing there and you want to remember it is the same, and the fundamental stuff inside, like due dates and reminders associated, maybe assignees. It's all the same. It's just a way of projecting.

So I think the – When I went to look at Wunderlist, I was leaving a job where I was dealing with massive scale and it was a multibillion dollar value business and people said, “Oh, you’re going to work on a to-do-list? Really?”

If you search for to-do-list on Google, you’ve got like hundreds of thousands of results for even a code implementing to-do-lists. It’s like the “Hello world” now for programming, but my ambition with them was to try to create the last necessary to-do-list app. So create a great integrated experience everywhere with an open API and intelligence behind the scenes. That's one reason we were excited about getting acquired, is we thought, “Well, even if we can't hit the entire internet, at least the canvas that we have available on the number of users that we can help at Microsoft is massive.” So we can actually achieve this vision inside Microsoft with integrating the Microsoft products and then still be open to the outside world, because Microsoft is like that these days. We have open APIs that you can connect to.

[0:30:11.9] JM: So eventually you shifted over to developer advocacy or other kinds of work within the cloud, as Microsoft is going through this big transition towards building more cloud products. I've done a lot of coverage of some of the new Azure products and stuff related to Kubernetes and containers and Cosmos DB, and there’s a lot of exciting products that are being built. So what is your role look like today?

[0:30:43.8] CF: So it just changed a few weeks ago. I actually moved initially into the Azure organization to lead something called Startup Advocacy, which is a new team that I created. The purpose is to help the startup world and VC world by association and to connect better and provide more value to that audience as a company. I did that because being a startup guy myself, I was kind of ready to get back into the startup life. It turned out I could find a job at this new big company that I still love talking to the audience and the people that I love all the time.

So what that looked like was the CTO's and the residents from all of the worldwide accelerators reporting to me and thinking about how we can help the CTO community, a startup CTO community both in the ones that are going through our accelerators and in our local ecosystems, so London, Berlin, Sydney, etc. We have accelerators called scale ups all around the world.

I have recently brought that team with me into larger group called Cloud Developer Advocacy, which probably many listeners will be familiar with various members of that team. The team was formed a little over a year ago, and you can think of it as the rethinking of developer evangelism at Microsoft.

So we hire people that are part of technical communities and already leaders on those technical community rather than hiring people that are aligned to our products and then having them try to go out to tech communities. For example, we have people in the node community. They're already deep in the node community. We tell them, "Keep doing what you're doing with the node community. That's why we want you. We want people that are really connected to developers and their needs in modern and heterogeneous environments. But as you do that stuff, help us make Azure a better place for those developers and help us tell those other developers that Azure is indeed a better place," when it is, and we try to do that rather than going out and selling specific Azure services on stage. We just work it in as the way we naturally do stuff.

So if you're doing a tutorial on – I don't know, an express JS app or something, you're deploying it into an Azure app service, or you're deploying at Azure functions, but you're not getting up and getting a product pitch for Azure. I lead that overall team. I'm still learning my way around, but the combination for me of developer community with my background and open source in the Ruby world and all the other stuff I've done with developers in the past startups and then cloud computing is kind of the perfect thing for my interest.

[0:33:36.0] JM: Well, if you look at – So one way, I think, that we could frame this of how much is changing right now, is if you look at what you were building with Wunderlist back in 2015, some of the problems that you had to solve relative to the technologies that you have available today and how you would solve them today. The products that you would choose, the open source technologies that you would choose will be so different today, and it's just been three years. I mean, the idea of having access to serverless or some managed Kubernetes solution, it's so different. Also, the kind of interoperability between you're kind of like open source, I am writing this code compute layer, and the managed services layer or the – Whatever, DynamoDB, or Cosmos DB, or name your API. Name your machine learning API, or your machine learning framework, or your hosted machine learning framework. All of these tooling has changed so

much in just three years. So when you look at that pace of change, how does that inform like how you're spending your day-to-day?

[0:34:59.2] CF: I will say I'm very excited about the changes that have happened, especially in containers, Kubernetes, that sort of area. I have had an almost career long obsession with trying to figure out how to build software systems that can live a long life, and I didn't really say it. But when we went through the re-architecture of Wunderlist, and I did the entire thing through that lens. It started with a talk I gave many years ago that I've done replicas overtime called legacy, where we try to recast the word legacy and software as a positive thing. Like it is almost everywhere else leading a legacy.

Through that, I developed my own set of biological metaphors for software, and I know this sounds like I'm not answering your question, but I'm almost there. The most important one is the concept of cellular regeneration. So you can look at a human body and just to totally oversimplify it. I am not mostly the same cells that I was when I was born. Somehow I'm still Chad and I'm still me and the system of me still survives, and compared to most software systems, it's pretty healthy, even though I don't take great care of myself all the time.

So that was one thing that I took as a central concept in building what I ended up calling immutable infrastructure at Wunderlist, which is tiny pieces that are disposable. Servers are disposable. Run times are disposable. Codes should be disposable. You should be able to throw it away and rewrite it, because it's so small and so well-decoupled.

We ended up at that time building a crazy amount of infrastructure around these concepts, and I say crazy like when Microsoft did due diligence, the team that they contracted with to do the really deep stuff came back and literally wrote over engineered as like the headline, but in a celebratory way. I was very proud of that, because it was – If you back up and say, "Wait a second. This is a to-do-list. Whoa! Why do you have all those crazy stuff?" I can see how it feels over-engineered.

But what we were building was a way of working with systems where we would never have to rewrite the thing again. We would rewrite all the pieces of it, and we did ultimately rewrite at least 80% of it, but we never had to, after that, do any kind of massive overhaul to the system.

So getting to today, we built a bunch of stuff that with the advent of Docker and orchestration layers and all that stuff, becomes much, much easier to achieve. The idea that run times should be temporary is kind of obvious now. But back then we were literally killing and re-creating VMs just to ensure that we had cycling and regeneration in the system for its health, and it was really [inaudible 0:37:59.9], because it is virtual machines and not containers.

So I'm excited about what all these stuff enables. I think we still have a long way. I call this an incremental step, all these change that you're talking about. It's still noisy, and I think we have a long way industry-wide to figure out how to adopt this stuff to great effect in the same way that we don't really understand why we're doing micro services across the industry. We're just making little services, but they're still tightly coupled and have all the problems of the monolith, but more problems on top in most cases.

[0:38:34.2] JM: So what about the serverless stuff? This is a conversation I have with a lot of people in the Kubernetes community. By the way, nobody knows what is going to happen with serverless versus Kubernetes. There is no like tech Oracle that you can go to and ask, "What's the future of serverless versus Kubernetes?" Nobody knows. There's all these serverless frameworks being written on top of Kubernetes and there's obviously these just serverless deployment functions on Microsoft, and Google, and Amazon, and they're event driven, and so on.

But what's your take on that? Where does this go and what does our state-of-the-art compute layer look like in – I don't know, three years?"

[0:39:23.0] CF: If we can avoid the human temptation to invent things ourselves, the right way to go is to let go of building stuff that isn't the core of our business, whatever that is.

[0:39:36.2] JM: Yeah, I agree with that. That's surprisingly controversial.

[0:39:39.9] CF: It shouldn't be. At Wunderlist, even back then, my rule was everything should be disposable. You can't actually change the configuration of a server and everything needs to be stateless so you can throw it away. The exception of course is stuff that stores data. So we

started moving to store all data in some sort of managed service. So we can have a server to manage it. So whether it'd be even RabbitMQ, Redis, all the databases. Those are all just managed hosted things, like RDS and Amazon.

I think that's the way to go. You need to get out of that business in the same way that decades ago people needed the trust assembler to generate good machine code, and eventually trust C to generate good assembler. The levels of abstraction raise, and if you really care about getting things done, you grow up as high as you can. I think, for me right now, that would be how I test the limits in the way that I was testing the limits with the Wunderlist architecture when I started. I would be testing how little can I actually write and how few services can I actually manage.

My opinion is that Kubernetes and all these things are going to be pieces of infrastructure that few people have to deal with and they enable "serverless" Nothings is actually serverless, but they enable this abstraction where you just deploy code that elastically scales, and I hope that people use functions, lambda, etc., in ways that map to their names. Meaning don't deploy a whole MBC type app into it and have that be something that gets managed, but try to constrain yourself to deploying small functions into it that are decoupled and ideally even stateless were possible. I think that would be better for the industry.

[0:41:36.4] JM: I completely agree with that. Unfortunately – Look, I'm not writing software today. So I'm not really in a position to say this authoritatively, but I'm not sure I envy the people who are doing the Kubernetes Nettie's migrations today, because I think – Okay, they're migrating to a hybrid cloud scenario where they're on a manage Kubernetes provider together with their on-prem infrastructure. Yeah, that's great. It's certainly an improvement, but I think in a couple of years the serverless conversation is going to be more at the fore and I think going to be more accepted and less edgy and people are going to be like, "Why did we invest in this infrastructure management layer? Why did we do that?"

[0:42:18.0] CF: Yeah, although if you do it right, you set yourself up for the future. If you do containers and Kubernetes all these stuff in the philosophically true way, you have a configuration that's really easy to port even it's by writing scripts to generate something else out of it and you hopefully have code that is in small decoupled components that make it easy to port as well.

So that's the way I would be thinking if I were doing a migration like that right now is, “Okay. This is my metal layer that enables me to very easily evolve to the next step. Then from here on out, it’s all just declarative.”

[0:42:57.1] JM: Yeah. Anyway, since I'm already speculating about the future and subjects I have no business speculating about, the last time we spoke we talked about some different investing areas. So we are talking about cryptocurrency infrastructure, other things around Kubernetes, developer productivity, machine learning and then the intersection or the power set of intersections of all of those subjects.

Give me the Chad Fowler investment thesis, the venture investment thesis. Give me the five minute Chad Fowler investment thesis.

[0:43:31.4] CF: Oh, I don't know if I can construct that, but I can take all of those words that you just said and tell you something about how I think about it. It may not be an investment thesis. So I am an investor. I am a venture partner with BlueYard out of Berlin, and when we were talking it was because – When you and I were talking about this is was because I was about to run a small conference in Iceland, which we did called Future of Software Development, where we got all these forward-thinking practitioners, researchers and entrepreneurs that are doing stuff to bring forward how we do software development in the future in a hopefully nonlinear way as well as people who have made historic contributions to the field previously.

I don't know if I can call it an investment thesis, but if you take that combination of those things you just talked about, so machine learning, big code as some of the machine learning for programs, synthesis people talk about it. So the existence of, for example, GitHub a public repository of repositories. This is historically something that has not existed and now it does. The advances in machine learning itself and the performance of machines that make stuff like deep neural nets possible.

Then cryptocurrency economies, and I say that because it's not just about making a coin, but I think when you do crypto currency right, it's because you are enabling a new protocol to be developed on top of an incentive economy. Lunge these things together, and I imagine a world

where, for example, open source development is really implemented on top of protocols that are driven by economic incentive.

So you could, for example, buy shares in an open source project and have that be part of how governance works, but also gain shares by contributing, and contributions could be baked into the protocol so that the protocol of how humans interact in open source projects incentivizes stuff, like helping people on forums and contributing documentation, not just contributing code.

So if you have that, then I believe we are getting to a place where software the actual code is not the value anymore. We're pretty much already there, but most people haven't admitted it yet. So therefore all software can be open source. Then you just charge by or you make money off a software by running it for people and providing safe environments for it to execute.

So also software is open source. Open source development is done on this token economy, and you can make money off the token economy. It's possible that software developers don't actually work at companies anymore, or at least there is a huge migration away from that. Instead they're just working on open source projects and they're getting paid for the work they do. Now add program synthesis and various other techniques that are machine learning driven approaches to learning – Computers learning to program computers. I could imagine things like, let's say, hard bleed comes up, for example. There is a race early morning for someone to be the first to release the bot that goes and plugs every open source project in the world. Suddenly they're rich in a day because they have solved a bit problem, or you find a class of security problem or class of even software quality improvement and you can unleash your code, your bot.

So I can imagine botnets that are created that live on top of this token economy that are really driving improvements and eventually even creation if you really want to get futuristic of new features by reading descriptions of feature requests. Yeah, all very futuristic and hand-wavy and unlikely to actually happen, but I think some versions of this will certainly take place in some components of this will get built.

[SPONSOR MESSAGE]

[0:48:01.6] 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]

[0:50:09.0] JM: I mean, I agree with pretty much all of that at the – I mean, the only thing you and I think would maybe disagree on is time horizon. I mean, we probably are both on the same page that we don't know what the time horizon is, but we can look at the foundations and say this stuff is possible. All of that stuff is possible. You and I could spend days mapping out on a whiteboard how it's going to take place in the different market pressures that are going to lead us in that direction. For example, it's all going to be open source and it's going to be developers hacking on the same thing, because look at what happened with Linux. Looked at what happened with Kubernetes. Looked at what happened with name your open source technology

that people centralized around and obviously other people came up. It wasn't just ReactJS. There was Vue.js sort of on the side, but Vue.js makes a very consciously different design decisions that give it its own community.

But by and large, developers do not like the idea that stuff gets written exactly the same for one company and another company. It's like, as engineers, it irks us, because it's just like this – If only all the developers were working on android or all the developers working on iOS. Just pick one. We don't need this duality. We don't need people rewriting everything in two places at once.

[0:51:30.3] CF: Although that is part of the way that open source works. Of course, the old Cathedral and bizarre thing, it's the bazaar model, and competition and open source communities creates better software. But you're right, that I think that same mentality of don't write this over and over again, software developers have usually to a fault. It will play a part in this, and it has already.

I mean, I remember I was working actively when Netscape was open sourced and when Eric Raymond released that Cathedral and the bazaar essay, and I was part of a team at GE that introduced open source software for the first time and really had to fight for it to be okay. Now, I mean, I even tell people at Microsoft, "Stop saying open source so much, because everyone just assumes things are open source now. You don't need to say that anymore." It's become just how we do things. But you're right, all these components of this semi vision thing that I laid out exists. I can even tell you not just how they would happen, but I have the email addresses of the people working on all of them too. In some cases, multiple different teams working on solutions in each of these areas.

[0:52:42.7] JM: But how do you know which ones are working on webvan and which ones are working on an Instacart?

[0:52:47.6] CF: That I don't know. As an investor, I can tell you it's all about the people. You talk to the people and you're blown away by people and you just follow them and you can help but invest. First example, I think it was the first investment BlueYard made was in Protocol Labs,

which is the team behind IPFS, the open source file sharing protocol, peer-to-peer. Then the upcoming file coin, a bunch of other interesting software.

When you talk to Juan, who is the founder of Protocol Labs, you will be convinced by the end. He will move you from skeptic to believer. That's why you invest, because even if he is wrong, he's going to do something.

[0:53:30.0] JM: Yeah. Yeah, and that's the other thing, is like this money doesn't get wasted. It's like even if you – Even if you put money into both Instacart and Webvan, you would be a winner. You lose to ask two X your money and you gain 100 X. So it's just funny. People will ask, like, "Are we in? Are we in? Is it too early to invest? Are we in the Webvan days or are we in the Instacart days?" It actually doesn't matter. You should put money in either way.

[0:53:58.7] CF: Yeah, or not. Just depending on your profile and appetite for risk, of course. But yeah, I'm with you on that.

[0:54:05.7] JM: Of course. Also we don't know, the time horizon could be compressed. Maybe there're a thousand Webvans before the Instacart.

[0:54:13.3] CF: Yeah, completely. Although it feels like it's more these days that things are speeding up. So there are probably far fewer Vebvans, or maybe there's a thousand in a very short period. I don't know.

[0:54:26.5] JM: Exactly.

[0:54:28.7] CF: But part of the conversation about time horizon for things changing in these radical ways in terms of how we develop software that influences me. I remember going to a YC Demo day a couple years ago, and I'm going to get it wrong, but I'll just say the wrong thing. It's right in principle. There was one team that was like 3D printing rocket engines and another one had cheap satellites that you could put in space. Then I talked to another company right around then, it was building basically like android for satellites. Suddenly, I had not heard about any of these kind of stuff. Td then suddenly I realized, "Oh, well. I can afford to put a satellite in space with my own custom software that I can hack on." That just came out of nowhere.

At the same time, we have autonomous cars and all these stuff that felt like it was “the future” or THE FUTURE. The future just showed up all at once. So I don't know. I might just be naïve, but I feel like we can't predict at all, and all I know is it's going to come faster than I expect.

[0:55:35.1] JM: Do you think – Okay, last super speculative question, then we’ll talk about music. So when I was at KubeCon in Copenhagen, I talked to a few people and I sort of made sure I picked the right people to talk to, because talking to – It's funny, like talking about cryptocurrencies or blockchains at a Kubernetes conference is a very easy way to end up a wallflower and be like, “Oh, that's the guy that's talking about blockchains. We already decided that that stuff is not happening.”

It's funny. It's crazy, this distributed systems conference where nobody's talking about cryptocurrencies. It’s just very strange. But do you think there will be some kind of overlap between the Kubernetes distributed systems, like centralized infrastructure community, and the blockchain crypto community, or is this like saying that, “Yeah. I mean, there's a overlap between Linux and Kubernetes in the sense that Kubernetes is used to manage Linux nodes.” Do you see an overlap there or do you think it's like not even worth kind of going in that direction?

[0:56:39.6] CF: I think if you start talking about product names, like Kubernetes, then maybe not. But if the question is, “Do “traditional distributed systems people” end up overlapping with the blockchain people and their interests overlap?” Absolutely. It’s just a matter of how long.

I remember trying to figure out what the hell Ethereum was when I first heard about it, and I actually heard a great podcast that you did where you explained it quite well in the podcast. But you hear people talk about Ethereum. The layman usually thinks Ethereum is just another cryptocurrency thingy. It's for money. But when you talk to Ethereum people, they say this is a distributed virtual machine and you run programs. They call them contracts, but they’re programs, and they’re programs that are immutable, and theoretically it cannot be turned off by anyone too, because they’re decentralized the way they’re run.

There are teams that are taking that topic and metaphor very literally and building layers on top of it. So there's like ZeppelinOS, which is if you think of Ethereum as a virtual machine, then ZeppelinOS is an operating system that provides services above the hardware layer.

Right now, we had a panel actually with Zeppelin and Parity some others at this future of software thing, and I said – I asked the question to the panel, “If this is a distributed virtual machine, it’s the slowest, crappiest this virtual machine ever. It has some nice characteristics in terms of all the immutability and guaranties in cryptography, but it just can’t run anything real. So is there going be a time that you can really run apps on this thing?

Of course, I don't remember the exact answers, but the answer was something like, “Yes, sort of.” But I think that day is coming where something happens that makes it possible to run things at scale and with decent speed across this distributed, say, in a cryptographically secured virtual machine and you get all the benefits that you know about now from it if you know those benefits, if you care enough, but it actually also runs in a decent way. When that happens, then the traditional systems people are going to have to take notice.

[0:58:59.2] JM: So when the economy becomes entirely open-source and it's flowing, the economic value is flowing in a distributed fashion, and we end up with the basic income where everybody can now just be creative. Robots are doing everything. Perhaps music will become an important thing for people to think about on a serious level as opposed to just entertainment value, because maybe we'll have more artists in the world. Really, a messy segue there. But –

[0:59:34.9] CF: No. It was great. [inaudible 0:59:36.2]. Thank you.

[0:59:37.6] JM: So the last coverage we had, we talked a little bit about Splice. Since then I actually did a show about Splice, which was really fun. I mean, I know you're a fan of the Splice people. Basically they are trying to solve music collaboration from all the different dimensions. Like you see all the things that you can do on GitHub. Say what you will critical of GitHub. A lot goes on on GitHub, and there’s a lot of different things. It's a really vibrant ecosystem, and it seems like Splice is going to be the same thing for music, or already is.

What do you think is the future of music? By the way, you went to school as – You studied music in school and you've thought a lot about this intersection between music and computer science. I think Splice is going to change things. It's going to make collaboration at scale possible in music. I think machine learning is going to change the life of a musician. You think about all the things that we have to do as a musician. If you're a multitalented musician, you've got to think of a chord progression, and you've got to do mixing and mastering and all these things we could very well define, like, "Yeah. This is what a good mix sounds like. Machine learned to recognize that and give it to me. Don't make me insert an EQ. That's just unnecessary." Yeah, we're still doing it today. So how do you think the life of a musician changes in 5 or 10 years?

[1:01:01.3] CF: I think it actually degrades in 5 to 10 years in certain ways, and that is because of the democratization of music. The things that you were talking about make it possible for people who didn't spend hours and years of their life deeply studying and practicing like I did to create things that sound as good or better. Even today's technology allows that. You can be a hobbyist and create something that ends up being a hit. There's no reason you can't, and not just something that's a hit in terms of like writing a song, but fully producing it. It's amazing, I'm on a computer here with Ableton Live on it and the stuff that you can do barely even knowing Abelton just blows my mind.

So I think the ability to live as a musician is a little harder. It's already getting worse and worse, has been for years. I think that continues, and maybe basic income is the answer to not completely lose the arts. The only saving grace there though I would say is that a lot of the work that musicians do is not art anyway, and they don't think of it as art. I say that as someone who has been a professional musician, and I have a lot of friends who are, I think machines can eventually do commercial music better than we can. There's no reason they couldn't. It's not art. It's programmatic. I say that as someone who – Like a couple of years ago, I moved back to the US after living in Germany and working on Wunderlist and moved back to Memphis where I had been a professional musician. I pulled the saxophones back out and started practicing and played a few gigs around town. Very friendly reception for my old musicians buddies who used to be the up and comers in Memphis, and they're now suddenly 20 years later the first call people.

But what I found is it's sort of a bummer. It's not really fun. It's a job for them. For me, I was excited to play a blues gig, but ultimately I want to do something that is completely creative. So it doesn't hurt my feelings that much that maybe kids won't grow and play a Mustang Sally 14 times a week on Beale Street in the future, because that really isn't art anyway.

I've talked to a couple of startups in the space of music synthesis through machine learning, and I'm excited about the future there weirdly enough. I think not only can machines more efficiently generate music than people when there are certain parameters, but they can do things that we couldn't. So, for example, if you were generating music on the fly inside a game, the engine could react to what's happening in the game. I imagine your character in a game has a theme song that always is identified with you as the character. As you traverse the game and do different things, you have different variations on your theme that convey the right emotion at the time. You can't compose something like that. It's not actually possible.

So I think a lot of interesting stuff will happen there, and I do think that ultimately music can get better, at least the ability to express one's self artistically can improve through all these automation in the same way we are talking about software developers. Build the core as a software developer. Not all the infrastructure. That doesn't really matter. Same thing is true as you were saying in music, that you spend so much time as a musician even when you're trying to artistically express yourself doing things that are not the core of what it is you're trying to create. So I hope that technology, and I believe that technology will rapidly fill gaps there and improve that experience.

[1:04:42.3] JM: Yeah. So the life of a musician in 15 years is like you sit down in front of your digital assistant and you're like, "So today I've have really been you know thinking about the relationship by head in elementary school and I've have also been listening to a lot of 9 Inch Nails, and some Frank Ocean and a little bit of Skrillex, and I really like B-flat today," and then you just swipe through different configurations of that, or that set of influences.

[1:05:11.2] CF: Yeah. I mean, it might even know already that you've been doing all those things, because the computer is watching you.

[1:05:15.3] JM: You're right. Yeah, exactly. "I've generated some songs for you."

[1:05:22.0] CF: I do that anyway. In fact, some of the best work that I've done as a musician has been because of constraints that were imposed. Like there's a site called Song Fight, where they just give you a title, and it used to be once a week, and you have to write and record and publish a song of that title.

In a sense, it's generating something for you there. I mean, at a very crude level, but I've even done stuff like auto generated lyrics with Ruby scripts 15 years ago and then try to write songs for those.

[1:05:56.6] JM: Oh, okay.

[1:05:57.3] CF: Auto [inaudible 1:05:57.1] generated song titles using character-based CNN by feeding in all that jazz song titles in the world and then see what that inspires. So I don't think all of these sets of ideas are very different from, "Well, let's just hear some random notes that might sound good too and see if that spawns an idea." I think these are tools that will use as creative people.

[1:06:21.3] JM: Chad Fowler, I think that's a good place to stop. Very abruptly, but I'm sure we'll do something else in the future. There's a slot here and we could've explored more. So thanks for coming on the show. It's been great getting to know you and I enjoy our conversations a lot. I like the free-flowing nature of them.

[1:06:38.8] CF: Yes. If we try to do anything else, it would absolutely fail. So I'm glad this is what we did.

[1:06:44.7] JM: Cool. Okay, awesome. I really enjoyed this episode. This is great.

[1:06:47.9] CF: Great. Me too. Thanks.

[END OF INTERVIEW]

[1:06:52.5] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested. With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers.

I know that the listeners of Software Engineering Daily are great engineers because I talk to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many, newer, hungry software engineers who are looking to level up quickly and prove themselves. To find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineeringdaily.com.

If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. Send me an email at jeff@softwareengineeringdaily.com. Thank you.

[END]