EPISODE 881

[INTRODUCTION]

[0:00:00.3] JM: Internet of Things is the term used to describe the increasing connectivity and intelligence of physical objects within our lives. IoT has manifested within enterprises under the term ‘Industrial IoT’, as wireless connectivity and machine learning have started to improve devices, such as centrifuges and conveyor belts and factory robotics. In the consumer space, IoT has moved slower than many people expected. It remains to be seen when we will have widespread computation within consumer devices, such as microwaves and washing machines and light switches, IoT computers have different constraints than general-purpose computers.

Security, reliability, battery life, power consumption and cost structures can be very different in IoT devices than in your laptop, or your smartphone. One technology that could solve some of the problems within IoT is WebAssembly; a newer binary instruction format for executable programs.

Jonathan Beri is a software engineer and the organizer of the San Francisco WebAssembly Meetup. Jonathan has a significant background in the IoT industry and he joins the show to discuss the state of WebAssembly, the surrounding technologies and their impact on IoT.

[SPONSOR MESSAGE]

[0:01:34.1] JM: Today's show is sponsored by Datadog, a modern full-stack monitoring platform for cloud infrastructure, applications, logs and metrics all in one place. Use Datadog’s rich, customizable dashboards to monitor, correlate, visualize and alert on data from disparate devices and cloud back-ends to have full visibility into performance.

Datadog breaks down the silos within an organization's teams and removes blind spots that could cause potential downtime. With more than 250 integrations, Datadog makes it easy to collaborate together and monitor every layer of their stack within a single platform. Try monitoring with Datadog today, using a free 14-day trial at softwareengineeringdaily.com/

datadog and they'll send you a free t-shirt. That's softwareengineeringdaily.com/datadog. You sign up and you get a free t-shirt. Check it out at softwareengineeringdaily.com/datadog.

[INTERVIEW]

[0:02:48.9] JM: Jonathan Beri, welcome to Software Engineering Daily.

[0:02:50.9] JB: Thank you. Pleasure to be here.

[0:02:52.8] JM: It's 2019. Describe the state of the Internet of Things market.

[0:02:58.3] JB: Oh, boy. It is an ever evolving market. It's past the initial hype cycle. It's past the early prototyping phase and it's starting to see a lot more definition in terms of use cases and companies really finding how IoT works for them.

[0:03:15.1] JM: Why does Internet of Things seem like one of those markets that's always five years away from happening and then five years passed and it seems yet again, it's just five years away from happening?

[0:03:25.7] JB: That's a good question. I think the final vision of a fully connected intelligent space is really easy for us to see, because of sci-fi, and we can immediately see the benefits of that world. The complexity of getting all those things connected and making sense of all those connected things and making it cost-effective, such that if the price goes to zero, to do all that is just so much more complex than we ever realized. As you go deeper and deeper into building out those core primitives, if you will, we're just so far behind than where we need to be.

[0:04:01.6] JM: You were at Google for six years from 2011 to 2017, how did Google's strategy around IoT evolve over that six-year period?

[0:04:11.6] JB: It started really early, even before I was involved with IoT, really with connected printers, from the earliest taking a device, adding connectivity and figuring how to manage it was in the cloud print team. That evolved into the Android and Android managed device, MDM

systems. I think it really kick-started when Google acquired Nest and started to have in-house expertise on what it takes to build a connected device.

Obviously, with the consumer angle, that's a whole layer of new types of use cases and challenges both technical and product market fit. That's really that inflection point. At the same time, Android was putting together their own strategy from their own learnings, which was started as project Brillo and eventually became Android things. Where we are today and I have been at Google for a couple years now, it in some ways looks like it's on the same path, but also I think it's pivoting yet again.

[0:05:05.3] JM: Let's see, you were a Nest when Nest was acquired, right?

[0:05:09.6] JB: No, no. I was at Google working on Android and Firebase and then I joined the Nest team shortly after.

[0:05:15.5] JM: Okay. What was your recollection of that acquisition? How did Google merge Nest and its IoT strategy into Google's own strategy?

[0:05:24.0] JB: I don't think the integration really happened until recently. That was somewhat by design, both from a Nest brand and the early founding team, the way they wanted to approach extending into more the consumer space and bring Google along. Also, I forgot to mention there was Google Glass at the same time, which had that same IoT connected problems to solve, but were a different team altogether.

When Nest joined, it was really going continue to doing, solve the consumer space to the best of your ability, beg, borrow and steal Google technology as it makes sense, but really it was its own thing. It has its own campus. It's about a seven-minute drive from Mountain View. It wasn't really integrated in the beginning.

[0:06:08.6] JM: As you've worked on IoT for so long, I assume you've identified some prototypical challenges of IoT platforms. What are the prototypical challenges of IoT?

[0:06:20.0] JB: Yeah. That's also a really good question. It actually is worth going into what is IoT in general, because I think different people will give you different answers. There's these axes, right? A Internet of Things, the thing that has internet connectivity. That's a general proposal, but it's so much more varied. Is a self-driving car, an Internet of Things device, is a smoke detector with a Wi-Fi radio, an Internet of Things device, using your smart phone; an Internet of Things device.

This probably goes back to your original question, why are we so always going to that pattern of five years out? It's because we're redefining and honing in that IoT is actually a lot of different things. An Internet of Things platform for self-driving cars, if such a thing exists, it's going to be different than an Internet of Things platform for building connected hot tubs, which would be different than building mobile platforms for phones and tablets. There are some general problems that you need to go solve if you're trying to build your own Internet of Things device.

Certain aspects you need to provide if you want to be a platform, to enable people to build those devices. We can definitely go into those in more detail, but do you have a device. Depending on the device, managing that device in terms of the software it's running, getting new software updates, keeping that device secure is actually very little to do with the Internet of Things, but is a core problem that needs to be solved first. That's actually what we think a lot of times with these platforms, especially device platforms.

Then providing connectivity for that device and there's a whole bunch of ways devices can be connected to talk back to a server somewhere, managing the devices from the cloud, so fleet management and doing that at small volume for a hundred sensors versus millions of devices and everything in between is another challenge. Data processing, machine learning, big data, that's another set of challenges and they can be more and more specific, depending on the category of connected device you're really talking about.

[0:08:17.6] JM: Let's talk a little bit about that platform idea. I can imagine different kinds of “IoT platforms.” You've got cloud provider platforms, like the Google’s, I think Android things platform, or their Google Cloud things. I don't really know what the messaging is around that at this point. You could have open source platforms. You could arguably say that the connected voice interface things, like the Alexa connections, that's a platform.

[0:08:48.0] JB: Yeah, totally.

[0:08:49.2] JM: What have been the different efforts at making platforms around IoT work?

[0:08:55.9] JB: Good question. I totally blanked around voice and really, that sometimes categorizes as an IoT thing. Though I always bucket those as basically smartphones with louder speakers, but I digress. Let's say, I build something connected. Let's say it was Alexa, Google home type of thing. There are definitely solutions out there that you can buy hardware from a cheap vendor and build out your own hardware solution and then take their software and maybe license some other software for the voice trigger words and NLP on device.

Then there are our cloud providers, like Google and Amazon, or even specific ones I think are popping up just for voicing and connectivity, to then wire it up yourself and so on to cover all those bases, as opposed to end-to-end solutions that may, or may not hit all those boxes. Maybe they don't provide hardware or whatever, but they are either being a general-purpose IoT platform, or specialized. Google's assistant program, or Amazon's Alexa voice program. That's a very small subset of what you might need to build your own connected voice assistant. Or you go to NXP and they have a hardware development kit that has a bunch of sample apps and reference code to talk to some third-party NLP servers. There is again that spectrum of what's required.

[0:10:08.7] JM: Has there been any effective efforts at open source ecosystems around IoT?

[0:10:14.5] JB: Yeah, there's been a bunch of successes and failures, depending on your metric of success and failure. There are big, big projects and little, small standards. I bucket those altogether. Oftentimes, you'll read articles about the standards war for IoT and how there's not really a war, but maybe there's just different opinions. There's a bunch of open standards that have been developed and redeveloped and we iterate on around things like the protocol to devices use to communicate to each other, or the security model for ensuring trusted communication between headless and low-power devices, or the way that they model data and say, "I'm a light bulb. Therefore, I could be turned on with a switch."

There's definitely that whole category of standards and open standards. Then there's the bunch of hardware, firmware, cloud back-end type of things. You can consider Arduino as an IoT platform and an open platform in some respects, because they have IoT offerings and they have connectivity as part of their solution. My former employer particle has an end-to-end IoT platform and their entire device and firmware and protocols are out in the open. Then there's the big players who are trying to get together and create consortiums to implement open source software that is referenced effectively. The OCF was one that was with a lot of big names in it and hasn't really progressed that far, but it is continuing to be iterated on. There are 20 I could ramble off.

[0:11:43.1] JM: The industrial Internet of Things has developed faster than the consumer Internet of Things. What kinds of applications have been successful in industrial IoT?

[0:11:55.7] JB: Yeah. It's interesting, because if you talk to people who've been doing industrial IoT for a long time, they would say, “We've been doing it for 20-30 years. We just called it M-to-M.” That is an industry term that’s sort for a machine-to-machine. Really, the distinction was in the 70s and 80s and 90s, they had connectivity, so sensors that cut to data and communicated even over IP networks, but just within a factory, or just between campuses.

It was the last mile of actually putting it onto a cloud server somewhere, that probably their distinction of why they say we were doing IoT forever. If those systems have been bought and installed, probably some of them are 15, 20 years doing this type of sensing and actuating and collecting data. Many of them are even over IP networks. As they bring them online, it's a natural progression. The use case is just-in-time delivery, predictive maintenance. If your machine has consumables, the refillable of those consumables, all those ideas actually came from industrial manufacturing, agricultural, and they've just been reimagined in the low-cost sensor, smartphone, microcontroller world that we’re just in today. They're probably the pioneers in some ways.

[0:13:10.6] JM: We'll get into a discussion of WebAssembly eventually. I just want to outline in more detail IoT before we can get to WebAssembly and intersecting with IoT, because you're something of an expert in the IoT space. Let's talk a little bit more about the security of IoT. I think security encompasses several different things. Security encompasses deployments and

software updates. It encompasses the connectivity and how you do permissions of connectivity and how you enforce passwords. We could always talk about Mirai botnets. Tell me what the outstanding problems of security and IoT are.

[0:13:53.6] JB: Security for an Internet of Things device is very similar to how we think about our smartphones and our servers, except the constraints are completely different, so we have to treat them differently. What I mean by that is you have data at rest in your device and maybe an end-user has accessed that device, or malicious software could be installed on that device, so you have to protect that data on the thing that you care about.

There's also communications, so there's possibilities for eavesdropping and men in the middle for sending data out. Maybe there's sensitive information. There's also downloading, the ability to interject and install a malicious software on to an Internet of Things device is very similar to smartphone exploits. Then there's the servers that they're talking to, right? You knew about data at rest and encryption, all those pieces.

The reason why it has to be reimagined is because these devices are nothing like the hardware on your laptop, or your smartphone and your server. Just for a frame of reference, a motion sensor that's probably in this room I'm in has 64 to 120 megahertz of processing power, as opposed to the quad-core gigahertz that's on my laptop. The RAM that's available is maybe 64 kilobytes, maybe a 128K, if they shell it off with a bigger component. The power budget is milliamps. A traditional, let's say certificates, like X509 certificates just wouldn't even fit on this device alone and be able to load memory. You have to take everything that you would be building for modern unconstrained devices and scale them, or reimagine them to fit on these more constrained devices.

[0:15:32.5] JM: IoT devices can store data locally on an embedded database. They could also shuttle it to the cloud. What are the patterns around what data to keep on device and what to shuttle to the cloud?

[0:15:46.0] JB: Yeah. It also will depend on the type of IoT device we're talking about. The motion sense in this room has no storage. There's nothing saved on this device. It might have some caching and RAM before it sends it off to some gateway nearby, but there's nothing

available. As we move up the stack and more capable devices and less constrained, so the gateway maybe it's talking to, maybe it has tens of megabytes of storage for configuration and local caching. I mean, in the future we'll start to see devices with machine learning models on them, so they can do inference before they shove it up to the cloud.

Then there are things like laptops that are doing very intense processing, but are being deployed in a car or something like that and they could have gigabytes. You can imagine someone doing a lot of local processing for two reasons. Honestly, in my career I haven't seen a whole lot of this either one. One is local caching for dealing with offline scenario. You assume an online connection all the time. Well, that's great in the Ethernet and Wi-Fi world, but anywhere else, that's not be the case.

Then once you connect, you send that over the wire, over the air. The other one is inference, or reducing the amount of data you send over to the cloud, because either A, sending more data uses more power. If you're a battery-powered system, you want to limit that. Or B, the cost of bandwidth is really expensive. Maybe you're a satellite, or cellphone, you pay per byte, so you always want to reduce that. Probably there's a subset of that too for privacy and not sending everything over the wire.

In most of the applications I saw at Nest and at Particle, people were doing very rudimentary things with data and really trying as best they can to stream it to the cloud where they'll do more processing and more of your traditional analytics.

[SPONSOR MESSAGE]

[0:17:41.8] JM: There are so many good podcasts to listen to these days and it can be hard to make time to sit down and read a full-length book. There are more good books than ever. I like business books and self-help books and history books, but I don't have time to get through all the books that I want to.

Blinkist gives you the best takeaways, the need-to-now information and the important points of thousands of non-fiction books, condensed into 15 minutes that you can read or listen to. I like Blinkist, because I like audio and Blinkist is an innovative useful audio format. You can get a free

7-day trial and support Software Engineering Daily by going to blinkist.com/sedaily and signing up.

On Blinkist, I've listened to a few very long books about China in their 15-minute form. I also use Blinkist to review great books that I've read in the past, such as Principles by Ray Dalio, or Being Mortal by Atul Gawande.

Try out your free 7-day trial by going to blinkist.com/sedaily and signing up. That's blinkist.com/sedaily. Get those books condensed into 15 minutes and get more throughput in your book reading activities. Thanks to Blinkist for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:19:20.7] JM: We've gotten a bird's-eye view of IoT at this point. I'd like to ease into a conversation about WebAssembly and then we can intersect the two. You organize the San Francisco WebAssembly Meetup. You've been interested in the space for a while. Why did you originally get interested in WebAssembly?

[0:19:39.9] JB: Yeah. It started with actually the IoT story, which I'll get to later. When I started getting connected with the community and talking to some of the thought leaders in WebAssembly, I asked them if there was a meetup that I could go attend, or maybe sponsor to my employer. Surprisingly, there was nothing in the Bay Area. There was a handful of JavaScript meetups that would have a session here and there about WebAssembly, or some of the blockchain meetups having a spin on WebAssembly, but nothing.

The reason why that surprised me so much is a lot of the core of the protocol and standards is developed in the Bay Area, between Google and Apple and Mozilla and Microsoft as well; the four drivers of the original spec. Three or four have their headquarters, or a large part of their teams here. Yet, none of it was being discussed in San Francisco. I took the opportunity to start a community and invite people in and see if there's any interest. It turns out there was.

[0:20:35.9] JM: Give me the state of WebAssembly from your point of view.

[0:20:38.9] JB: Yeah. In 2018 when I started talking about it, the majority of folks were either at these big companies doing spec development and reference implementations. The rest of the community was a lot of front-end people who are always keen on the next technology and tried it. They did the hello world, they did the add two numbers thing, which is early days, despite the fact that it was fully deployed in all major browsers without any flags. It was ready and primed for people to start adopting.

Now in 2019, the middle of the year, not only are we seeing major companies deploying in production, features like compression, or transcoding, reporting full applications, or seeing jobs that are WebAssembly engineer in the title. That's one of those, I guess milestones for technology readiness. Is it a thing that every developer will be using? Is it going to replace React? Certainly not. It's definitely not there if that was the case, but it's no longer a toy and something for academics, but a thing that is now a tool in the toolbox for many companies.

[0:21:48.2] JM: What does it mean to be a WebAssembly engineer today in 2019?

[0:21:52.8] JB: I think it means you wear two hats. There's a website and Twitter account, WebAssembly jobs and I check it every day. A lot of the places who are hiring these engineers and I don't know if they actually use that title, but effectively it's what they're looking for is you're a front-end engineer, so you understand the domain of building something for the browser, but they need to also become an expert in some other language and technology.

A bunch of companies who do audio processing are looking for someone with C++ and audio expertise, but has also has done a lot of front-end development, or is front-end curious. I saw an article this morning with Ableton showing a demo of something and the tutorial which is – I mean, it's going mainstream in that regard. Or the physics engines, the game engines, unity, front-end development tools. Qt from Trolltech, they are heavily invested in WebAssembly and are constantly hiring folks to expand their capabilities to build desktop-like applications on the web their stack.

It's an interesting role I think, because they require this expertise in a technology that's only a few years old, plus all this other domain-specific knowledge that they're hiring for anyway before. Hard to hire, I think, but it's interesting inflection point.

[0:23:05.8] JM: Well, now I must find out what's going on at Ableton. I'm very interested in these digital audio workstations and how they're getting them into the browser. How is WebAssembly going to impact internet users?

[0:23:18.8] JB: The canned answer is most internet users won't even know that WebAssembly is making their lives better. I believe that and we're starting to see some of that. I think for users on mobile, it might have the largest impact in terms of battery memory, because they will enable applications that are rich user interfaces, that do processing, do that in a very efficient way on the device and send less state over the cloud.

Take your example of I'm uploading something to a Dropbox-like site. I need to gzip it and maybe even encrypt it. Today that's a very expensive operation to do on a mobile browser, especially on low-end phones. As WebAssembly becomes available on to those devices and many of them have it, now you have smaller files into the wire that are compressed faster.

[0:24:05.8] JM: The WebAssembly tool chain is the set of tools for compiling and running WebAssembly code. Give an overview of the tools in the WebAssembly tool chain. I realize it's very dense space and there's a lot of interchangeability, but give me your perspective on that tooling today.

[0:24:23.8] JB: Yeah. That is a very dense space in some regards, but also pretty shallow in other areas. In the sense of what's different than the normal technologies we're familiar on the web is that WebAssembly is always going to be somehow compiled, whether it's compiled by the developer, or compiled by the runtime, there's this compiler toolchain aspect to it. There's a bunch of folks working on different types of compilers and taking one type of compiler and reimagining it, so it's a little bit easier, but there's that aspect of taking the one language and compiling it down to WebAssembly. How you debug that? How do you build a very complex application with that? That's one area where it's getting more sophisticated.

There's the language-specific aspect of that development. If you're a Rust developer and you're trying to get your Rust thing into a browser, sure there's the compile toolchain, but there's a whole bunch of libraries and supporting tools that the Rust community is developing around

WebAssembly to make that interface and interoperability better. Then there's the runtimes themselves. Runtimes is an interesting topic, because we know of the runtimes that we're familiar with are the browser runtimes and they all have WebAssembly runtimes these days, but there's the debugging of those in the runtime. There's the portability and even moving those out of the browser and embedding those into other systems, which I think is interesting and challenging.

I think, there's the interplay between WebAssembly and let's say, the host environment. I'm using general terms here. When you run your WebAssembly code in the browser, there's less about the debugging about it, but actually how does WebAssembly interact with let's say the Dom, or JavaScript. If it's running outside of the browser, how does it interact with the operating system, or the hardware underneath it?

[0:26:04.0] JM: What's the developer experience today if I want to do something in WebAssembly, or what would I do in WebAssembly today as an average developer?

[0:26:15.0] JB: One of the oldest tools and the oldest end-to-end toolchain is in scripting, which is both the compiler, if you will, for not a very accurate term, and the set of libraries that gets compiled into it, such that it can spit out some JavaScript that you can just drop into a page. If you look for a hello world WebAssembly, that's probably the tool that's going to be recommended. It takes your C++ code, or Rust code, converts it through the compilation process into a WebAssembly.wasm file. It will also generate you some JavaScript, which is just normal JavaScript APIs today, conscript module importing in the future. Now you have JavaScript APIs in your web application that you can call.

A lot of it is – the most common app is adding two numbers, and so you could write some C++ code. At the other end, you get a JavaScript Dom API, which is your own add these two numbers thing.

[0:27:08.4] JM: That actually illuminates the point that the WebAssembly toolchain today is mostly assuming an environment where there's a JavaScript engine and a browser-based environment, or perhaps a headless browser, or somewhere where you have a V8 JavaScript

engine, or the SpiderMonkey JavaScript engine. Does a WebAssembly toolchain always assume that there is a JavaScript based environment around it?

[0:27:42.1] JB: Great question. The answer is no. I think you've had guests on the past who talked about what's in the spectrum of assembly, which is non-web embeddings. The standard actually defines the core and the JavaScript later and the implementations that exist, they have those separations. That was for future looking of applying WebAssembly outside of the browser. JavaScript and a JavaScript runtime is only one place it might find itself. One of the newest efforts within the working group is the concept of well, if we run it outside the browser, what are the things that your WebAssembly code might expect or need to operate? That's the WASI, the WebAssembly Standards Interface and some other related efforts that are going on.

[0:28:23.5] JM: WebAssembly provides some sandboxed isolation, which offers some security benefits. Describe the isolation model of WebAssembly.

[0:28:34.3] JB: I'm no expert in that. Probably, someone from Mozilla and Google could do a better job. The genesis of WebAssembly came from the web. Our browsers from very early days, had this idea of isolation once we move beyond tabs. Different browsers have innate isolation mechanisms, even V8 I think has they call it isolates. Other browsers have other ways to isolate. Because when you're on a webpage, you don't want it to reach out and grab cookie, or other information from another tab on your machine. All APIs that the W3C works on keeps that in mind and ensures that.

With WebAssembly being born out of that same working group, the TC39, from JavaScript, they borrowed a lot of those concepts of isolation and not assuming that there'll be good actors on the page. The JavaScript WebAssembly runtime, so the browsers leverage the existing isolation provided by the browsers we're talking about. The APIs that are going through the committee, there's all this – even a discussion as part of that standards process where they talk about how do we ensure the isolation model can be enforced by the runtime and then expect out in the specification itself.

[0:29:46.5] JM: I realized that some of this material might be a little too deep in the weeds, but WebAssembly, it's such a big space and I continue to be perplexed by it and the sheer volume

of things within it reminds me of when we started covering cryptocurrency related topics, just because of the sheer scope of what the ecosystem represents. With that in mind, I'm going to ask you just a few more lower-level questions then we can get back to some higher level systemic questions around WebAssembly and specifically WebAssembly and IoT. There is this WebAssembly system interface, which as I understand is defining the interface between the WebAssembly world and the file system, the operating system, the how are you allocating space for files? How are you accessing files? How are you accessing lower-level system resources? This is pretty important if we want to have a consistent way of creating these non-JavaScript abstractions, these language neutral abstractions that we can deploy over the internet, execute in a safe fashion and access lower-level system resources. This is a really useful idea. What's your understanding of the WebAssembly system interface?

[0:31:08.2] JB: That is generally in the direction of its instated goal. It's interesting, because when we – coming from the web and think about web technologies, there's actually always this assumption that we took for granted. That there is a host environment, which is the browser. It does far more than just run your JavaScript code, or render Dom elements. It plays the role of a host, or literally the operating system. There are primitives that are there as part of accessing networking, accessing file systems, and also primitives that are missing that you might expect from a traditional host, or operating system.

When we transition to not the browser, we have those gaps, right? The language, the runtime is expecting certain things to be there. WASI fills in those gaps that we actually just take for granted in JavaScript, that any language that needs to be running in a host environment, or in a non-host environment would expect. A lot of languages have specifications. Sometimes they're called free-standing mode, or host list mode, so C has this in the language. It doesn't expect, or assume there's an operating system under the hood. This brings a little bit back to IoT.

Some languages make that assumption. Some languages specifically call out, “Well, if we don't have an operating system, then we don't have a file system, or we do not have networking, or may not look like a UNIX network socket, etc.” Rust also has some of these primitives. What WASI is trying to do is to find that gap between a language that may have a free-standing context and doesn't need that, as well as languages that just have that built in and require that

to run in the first place. From there, actually doing the things that you might want to do in a host environment, so running to file systems, opening up sockets, etc.

[0:32:54.2] JM: The drumbeat that I've maintained throughout most of the WebAssembly shows is that the main thrust of using WebAssembly is so that you can send, for example a Rust module over the internet and execute it within a browser. This can be really useful, because Rust has certain language benefits that JavaScript does not. That's great if we can have non-JavaScript code execute in a browser-based environment. As we start to talk about IoT, IoT devices they already can execute code that's not JavaScript. We can execute whatever code we want on a Raspberry Pi. Why do we care about WebAssembly when it comes to IoT?

[0:33:45.0] JB: Answering that question is what got me interested in WebAssembly in the first place. Going back to three years ago, I was working at a company called Particle, I mentioned that before. I was responsible for the product, which is the embedded operating system and the toolchain and all the things around it. The lingua franca for IoT devices, if they're – they're the smallest microcontrollers with that 120K of ram to even Raspberry PIs is typically C and C++ and that's because of the constraints on resources.

Maybe as you get higher and more expensive devices, like a Nest camera, you might have the luxury to have Java, or other higher order languages, but really, you got to squeeze as much juice out of those devices as you can. A lot of the people trying to build on IoT, on these smaller devices are struggling. Struggling because there's three things they have to do; one, they have to build a connected product, which is what their end-goal is. Two, they have to use a language they may not be experts in, or being able to perform well on. Three, they have to also understand the constraints of that device.

One of the top requests I kept on saying is, “I want to build a device using JavaScript.” I don't know, take JavaScript and run on a microcontroller, because it's familiar, I know to write that well and test it well.” The problem is some of those early JavaScript runtimes used up the entire resource of the device before it even ran your code. Multiple languages just couldn't really run well on embedded system.

Then on top of that, let's say you could and they're actually now advances in performance, JavaScript, Python and Rust on these types of devices, the platform provider, so Particle in this example, would then have to write some interface between JavaScript, that JavaScript runtime specifically, and the lower-level hardware drivers. To turn that LED on or off, there's very specific protocols you have to communicate over, or talking to the register of that system in this world of handcrafted runtime. This JavaScript runtime is different than that other one, would require the platform provider to keep on rebuilding their device drivers and their network stack and their crypto stack to some degree, because each one is a unique snowflake.

When I started here about WebAssembly, there's a couple things that checked off the box. One, it was this ability to run multiple languages with a single target. It also meant that as a platform provider, we would just have to build a WebAssembly module that talk to a hardware or not, a JavaScript module, a Python module and the Rust module. I think the most interesting thing at the time for sure was that all these browser vendors were behind it. That meant the tooling, the debugging, the community of creators and people who understand it would grow a lot faster than any of these smaller projects to do a high hand-rolled high order language.

It's certainly an aspect of accessibility. I think innovation too. The ability to get a bunch of people who may not speak – can code in C and C++ well. Having tools that are more capable in can inspire new ways to solve these problems. Then as I got more into it and understood the space more, there's other benefits which are just super interesting, like the ability to stream a WebAssembly file and [inaudible 0:36:57.5] is actually – it's interesting and you can't really do that in pre-compiled C++ released at the operating OS level today.

[SPONSOR MESSAGE]

[0:37:14.1] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust.

Whether you are a new company building your first product like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals. Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer.

We've also done several shows with the people who run G2i, Gabe Greenberg and the rest of his team. These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack and you can go to softwareengineeringdaily.com/g2i to learn more about G2i. Thank you to G2i for being a great supporter of Software Engineering Daily, both as listeners and also as people who have contributed code that have helped me out in my projects.

If you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW CONTINUED]

[0:39:06.1] JM: Let's go deeper into that. If I have a WebAssembly module that's written in C++ and I compile it to WebAssembly and then I send it over the internet to be executed on a remote device, or in a browser somewhere, why am I able to execute that in a streaming fashion? I mean, because we know that if I was to pre-compile a C++ module and I was not compiling it to WebAssembly, then I would need the full module. I would need all of the bits in order to execute that. Explain the streaming compilation feature of WebAssembly.

[0:39:43.2] JB: Yeah. It's also important, a lot of the times when I talk about these devices, I think about those tiny embedded devices more than I think about the Raspberry Pi. The value gets gray, or when you get to something like a gateway. You take your typical sensor, or your door lock, they are running what's known as real-time operating systems. The real-time aspect is less interesting, it's more about how they're developed and how they operate. They're actually

monolithic operating systems as opposed to modern Linux, or Windows, or you have your operating system boots up and then you install user applications, your real-time operating system as they exist today are all compiled into one single binary. They're clever hacks of chunking up operating systems and allowing you to update a very specific region of memory, but those are fragile and those are really just used for over-the-air updates, not for runtime.

It would be really hard to break up that application. Now with something like one of the frameworks that do embedded higher order languages, like micro-Python is the Python for microcontrollers. They have a way to reserve a region of memory where the micro-Python runtime, the operating system boots up first and then it can call in, load in that Python file.

That is more like what we think about modern operating systems and it's very specific to micro-Python. That's not reusable in another environment in each language, for embedded systems and IoT have to reinvent how they do that. If you want it to have multiple languages to support that loading the application afterwards, it's really hard. Actually, it's really hard to do that in C++, because the way that binary gets outputted and loaded dynamically breaks down. As opposed to at least directly, since very few people are doing it, the WebAssembly load time is the thing that boots up.

Therefore, the WebAssembly runtime, the operating system, all the devices that are managed by that embedded operating system are ready to go and now they can load in WebAssembly modules as needed.

[0:41:36.9] JM: Is the positive impact of WebAssembly on IoT, is that becoming a reality today, or are we still very much in the pre-realization phases of WebAssembly coming to IoT?

[0:41:53.3] JB: Definitely in the early, early phases. Even if you ask me in January, I would say no one is really doing anything with IoT WebAssembly. They were our – the edge cloud computing, like what Cloudfare and Fastly are doing, which actually a lot of the use cases are IoT devices, but it's just your traditional serverless compute. There's two areas, which I'm personally excited about, which are just starting to be experimented with. One of them is running WebAssembly directly on a device.

When WebAssembly was going through the ratification phases, there was someone on Twitter who I'd mentioned Brendan Eich saying, “If WebAssembly is an instruction set in ISA and microcontrollers and other devices are custom ISAs, would it be a good idea to build a microcontroller, or a microprocessor that is natively WebAssembly?” There's a fantastic thread back and forth between Brendan and that person and basically know.

I think it was interesting question and idea and maybe someday will make sense to do it, but really running WebAssembly on those devices is starting to be experimented with. We talked about multiple runtimes and how those runtimes can exist out of the browser. Earlier this year, someone was able to take one of the C++ runtimes and port it to a very popular microcontroller, so you could run a WebAssembly on this device, which had gobs of resources, but it was a proof of concept for sure.

Just maybe a few weeks ago, Intel published a project that they've been experimenting with that is more complete implementation of a runtime that is meant to use the least amount of resources. It is both good for microcontrollers and other environments that would want to run WebAssembly. Intel is very involved with Zephyr, which is Zephyr OS is a real-time operating system that they built from scratch and they have a port of it running on Zephyr. They're knitting away and getting WebAssembly running on Zephyr, which is running on an IoT device. It's very, very early.

The second area, which I think is interesting, which we really didn't talk much about yet, is the protocols themselves. I mentioned IoT has lots of standards and standards for devices communicating to each other in different ways they do encryption to handle the idiosyncrasies of these devices. The majority of those are implemented by the spec writers in C++ in an academic environment. They're very hard to reason, if you're not an expert in that protocol. They're very hard to integrate into your product, if you're just a Go developer, a JavaScript developer. You see people trying to take those IoT protocols and make them work in other environments.

I think one interesting use case would be if those specifications were implemented either initially in WebAssembly, or just having that as a target, so that we don't have to keep on re-implementing those very specific and unique IoT protocols, just do it once.

[0:44:48.3] JM: Okay, so just to reiterate, what are the problems of IoT that we believe WebAssembly might be able to solve?

[0:44:56.9] JB: Yeah. There's the making it easier for more developers to write software that runs on IoT devices. Not the C, C++ and very memory managed complex type of applications, but more JavaScript and Rust and Go; making it easier for those devices to have multiple applications streamed to them. Then the third thing I would call out is implementing IoT protocols, which are very complicated and written rolling by the spec authors and make that a reusable implementation through WebAssembly.

[0:45:27.7] JM: Okay, beautiful. You touched a little bit on this earlier, but do we expect WebAssembly to change the hardware standards, or the hardware requirements of IoT?

[0:45:41.1] JB: I don't know. I think I'm personally optimistic and interested in it. Like that early tweet conversation with Brendan Eich, it is mostly academic. There's not a whole lot of value today that can be generated from let's say, creating an application-specific IC, or an ASIC. There are potentially security benefits. I was talking to the folks in Mozilla just about this actual thread, and the way that hardware security is handled today, it's usually general-purpose. There's a lot of companies trying to build more hardware-based security into their chips that are again, general-purpose. Maybe they're accelerating certain algorithms.

The isolation that we talked about and the streaming compilation, if there was a hardware accelerator, it means inside of chips that it actually make it a little bit better and more secure for a device to be running WebAssembly. Because these devices don't have this concept of isolates and untrusted code, they have a more – if you have access to programing, you probably are my master mentality.

[0:46:42.7] JM: How do you imagine WebAssembly impacting the cloud provider businesses?

[0:46:47.6] JB: We're seeing a little bit of that with Cloudflare and Fastly, creating their WebAssembly environments for serverless compute. I think we'll see more of that, certainly in the layer two providers, especially because they can offer compute with less servers. Maybe it's

because you haven't your own Colo and it's too expensive overhead, but you can get a local provider that just spins up your own dedicated space. As more and more types of computing problems, look-alike serverless, I think a lot more people are going to experiment with it.

People are going to continue to do crazy things in the cloud with it. I just got a demo. BigQuery, apparently can run WebAssembly, because BigQuery runs V8 and you can inject JavaScript and you can load a WebAssembly module inside your BigQuery and use Rust to compute stuff, which is just crazy, right?

[0:47:42.4] JM: Why would you want to do that?

[0:47:44.9] JB: I think mostly for academic reasons. You could reimagine a analytics big data processing system that allows you to do maybe in-browser compute in a language or framework like Python, or Panda, or something locally. In addition to the cloud processing, I think that actually makes more sense to do that using WebAssembly. Yeah, doing it in BigQuery more, because they could, as opposed to it was a good idea. As JavaScript has been showing up in more places, I think WebAssembly being part of that same standards body might make it interesting and surprising where you can run it.

[0:48:19.8] JM: Well, that's an interesting idea, because I've always been curious about the federated computing thing. This is SETI at home all over again, but my browser on my computer, it's got a bunch of spare cycles that could be used for productive things. It would be great if it could be constantly munging my personal data and providing me with recommendations and information about cool things. Wouldn't it be great if I could use those extra clock cycles and maybe WebAssembly can help me maybe make an extra buck or two? Maybe I can use the federated computing model to process data from other people, but that dream has been a long time coming, so I won't get my hopes up.

[0:49:05.2] JB: Totally agree. A lot of the non-IoT use cases, which is crypto and blockchain like you mentioned, they're all going down that path. Surprisingly, the largest user of WebAssembly outside the browser are a lot of the blockchain companies. A theorem was the first to make this stride with E WASM. Last I heard, there was 10 or 11 blockchain technology companies, or not, using Ethereum, implementing their own WebAssembly runtime or borrowing someone else's.

Part of the reason to do that is because it enables them to have more flexibility in the contracts that people can write, but it also allows them to do things in the browser. In ways where maybe for privacy or security you want to process it before it goes out, becomes really interesting for them. I had a thought experiment with someone – one of these companies after the meet up with a bunch of beers in our hand about well, what if WebAssembly being both a binary format and a way to do compute could for example, download a blob. Only I as the owner of that data can access it, decrypt it all within the context of my isolated browser session, do something with it and then send it off.

Or conversely, what if I was able to provide limited access to my very private data to someone? The key was given to them, such that they can only decrypt some of that data. Now you're talking about privacy with data that's truly portable, with air quotes and sci-fi around it. It's interesting, because before this way of thinking, there's been a lot of experiments with using JSON, or JavaScript, or whatever to send data and in a cryptic way and keep it really secure. The fundamental problem though is once it's at rest, once it's in the hands of the person who has it, at some point, they just need to unpack it and they can copy it and they can set it off somewhere else.

This idea that WebAssembly could actually be running its own code for the decryption could have more control over how it's being used and reused. I don't know. We've got laden and we ran out of beer, so we can go further with it. It is an interesting potential shift in paradigm, which I think would be cool.

[0:51:14.6] JM: Yeah, hard to know if WebAssembly on Ethereum is one of those things that looks like peanut butter plus chocolate, or actually is something as amazing as peanut butter plus chocolate. Is this just another added layer of false hype, or this really the key to executing all kinds of code on top of solidity, because solidity is this Ethereum language that is reportedly not that fun to program in. Anyway, I guess we'll wait to see what happens there. What other changes to the business landscape do you expect WebAssembly to lead to?

[0:51:54.4] JB: Let's see. I think it will certainly change the way to some degree and not going to be all fanciful about it, but change the way that people think about building applications. Even

we talk a lot about the browser and how the browsers can have these sub-sections of applications that are doing more complicated compute, or cryptography, or whatever. I also think it might even change the way people think about sharing code between the browser, the server and let's say, even native apps.

We saw a little bit with that when node and some of the front-end frameworks started to do some – the isomorphic coding never really delivered. I think with things like audio and audio processing, or DRM cryptography IoT, when the things you're writing ones that are keep on increasing in complexity, you might want to really reuse that now. As opposed to having, I don't know, your expressed JS routes, map to your React routing framework, that's a problem that you could solve other ways. I think this increasing complexity at the client would benefit a lot.

You might even see people rethink how they structure their applications. A lot of the teams I worked on, especially on the native side and Nest this a lot and other teams at Google, would take the most complicated part of their protocol, their security, even the ways they treat user data and do that in C and C++ and either compile that out if it's compatible, or embedded in their Android, iOS and server-side language.

I think it's going to come from that mindset and maybe more companies will do that as they have those type of problems. Or maybe they'll use libraries that are written by someone else who's actually under the covers using the same reused code. There's a bunch of, I mentioned audio a few times now. There’s a bunch of audio companies that are doing that. I don't know what Ableton is doing, but we should go check it out later.

[0:53:46.2] JM: Yes, we should.

[0:53:47.2] JB: There's a company called Superpowered and they do audio processing, voice chat and they have been investing a lot in WebAssembly, because they can reuse this very complicated and research-backed processing algorithms in the same context. I think people might just use their library and call it done and not even know it.

[0:54:06.2] JM: Jonathan, thanks for coming on the show. It's been really fun talking to you.

[0:54:08.6] JB: Yeah, thanks. It's been a pleasure.

[END OF INTERVIEW]

[0:54:13.9] JM: Podsheets is an open source podcast hosting platform. We are building Podsheets with the learnings from Software Engineering Daily. Our goal is to be the best place to host and monetize your podcast. If you've been thinking about starting a podcast, check out podsheets.com. We believe the best solution to podcasting will be open source. We had a previous episode of Software Engineering Daily where we discussed the open source vision for Podsheets.

We're in the early days of podcasting and there's never been a better time to start a podcast. We will help you through the hurdles of starting a podcast on Podsheets. We're already working on tools to help you with the complex process of finding advertisers for your podcast and working with the ads in your podcast. These are problems that we have encountered in Software Engineering Daily. We know them intimately and we would love to help you get started with your podcast.

You can check out podsheets.com to get started as a podcaster today. Podcasting is as easy as blogging. If you've written a blog post, you can start a podcast. We'll help you through the process and you can reach us at any time by e-mailing help@podsheets.com. We also have multiple other ways of getting in touch on Podsheets.

Podsheets is an open source podcast hosting platform. I hope you start a podcast, because I'm still running out of content to listen to. Start a podcast on podsheets.com.

[END]