EPISODE 1196

[INTRODUCTION]

[00:00:01] JM: Embedded software engineering is the practice of building software that controls embedded systems. That is machines or devices other than standard computers. Embedded systems appear in a variety of applications, from small microcontrollers, to consumer electronics, to large-scale machines such as cars, airplanes and machine tools. iRobot is a consumer robotics company that applies embedded engineering to build robots that perform common household tasks. Its flagship product is the Roomba, perhaps one of the most well-known autonomous consumer robots on the market today. iRobot's engineers work at the intersection of software and hardware and work in a variety of domains from electrical engineering to AI.

Chris Svec is a software engineering manager at iRobot. He started his career designing x86 chips and later moved up the hardware-software stack into embedded software. He joins the show today to talk about iRobot, the design process for embedded systems and the future of embedded systems programming.

[INTERVIEW]

[00:01:07] JM: Chris Svec, welcome to the show.

[00:01:09] CS: Thank you very much, Jeff. How you doing?

[00:01:10] JM: I'm doing great. You work at iRobot. Explain what iRobot makes.

[00:01:16] CS: Sure. iRobot, if you've heard of us or if you've heard of our product, we make the Roomba. We are the home robot vacuum – Or sorry. We are the home robot company, but our most famous product is the Roomba, which is a circular robot vacuum cleaner. It's certainly the one thing that people think about when they think about our company. We have a

variety of other robots including mopping. And over the years, we've done – We've actually been around for 30 years. We've actually done a bunch of other robots in the security space, in the remote presence space and done toys. We've done a lot of stuff along the way. But right now we are the home robot company and it's all about robots that help people at home.

[00:01:53] JM: What are the canonical challenges of building software for robots?

[00:01:58] CS: The thing that I like to say is that not even building software for robots, but just getting a robot to work. It seems like, "Oh, getting a robot to clean a room. How hard is that?" You spin a motor. You create a little vacuum. You suck up some dirt and it seems like something you could hack together in a weekend. And hacking together something in a weekend or a week or a semester or whatever kind of a project you have in mind, you can certainly get something moving with wheels and with motors and kind of bumping around your house and doing a good job or a good enough job on a room or two in your house. And that's the first eighty percent – Or that seems like that's what's the first eighty percent of the problem.

And then I like to say then the next eighty percent is getting that to actually work reliably in your house all the time, and the third eighty percent is getting it to work in your house and your friend's house and maybe your parents' house and a few other houses and then the fourth and fifth and sixth eighty percent end up scaling that up to be something that works reliably in every house all around the world no matter what floor surface people have, no matter what wall configuration people have, no matter what chairs people have or what weird coffee table heights people have. Just to get it working in the tricky thing that is the real world is kind of the hard part. And that's any robot. That's any system that has to work in the real world around the systems that we fill our homes with.

[00:03:15] JM: Why don't we see more widespread use of robots in the everyday world?

[00:03:20] CS: Robots are hard. Robots are hard. Robots are expensive. We've been working, iRobot – First of all, I should say I don't speak for iRobot, but I've worked there for seven years. So I can say a number of things about it. We've been working at our robot for – The Roomba

came out in 2003 I think, and we've been working on it for a long time. And the Roomba is good. It's a very good robot, but it ain't perfect and we're constantly making it better. So it's not an easy problem. And the economics are such that it's not an inexpensive problem either. It is not inexpensive to have a research team, have an engineering team to make these things. And it is not an inexpensive device to manufacture. And it is not therefore an inexpensive device to sell.

I mean our robots have a variety of price points, and I mean I'm not going to comment on the economics of that, but you're not going to find a five dollar robot that does anything useful for you. And depending on the market penetration and how much disposable income people have, there's supply and there's demand, there's engineering difficulties, and that's a whole other conversation. The whole conversation around COGS, which is cost of goods sold, which is kind of drives a lot of the economics and the realities of the consumer electronics industry, which is if you're talking about robots in the home, then you're talking about consumer electronics, which has a lot of similarities to your Fitbit or your smart watch or your smartphone or anything like that.

[00:04:46] JM: Why was the vacuum such a good use case for modern robotics?

[00:04:51] CS: The story, as I've heard it, at iRobot, is that we were doing a lot of government contracts back in the early 2000s and few engineers decided to just try this thing. And I think they actually did it without a vacuum at first. That's what I heard. They just used a sweeper, basically a brush. Kind of like the sweepers that they use to clean movie theaters, if you remember what movie theaters are, where there's no vacuum, there's no suction, there's no power, but it's just like a little manual brush or maybe a powered brush that spins and pushes dirt into the sweeper. And the story that I heard is that they got that to work and then they showed it to some different focus groups and just asked people like, “What do you think of this? Would you buy this? How much would you pay for this?” And then people said, “Yeah, but I'd pay more for it if I heard it actually vacuuming, if I heard it sucking up dirt basically.” And so I have no idea if that story's true. I've heard that from a couple of old timers at iRobot and I saw it in some recollections somewhere online I think. But that is sort of the genesis of

half accident as to how the Roomba was invented and came to be. And then turns out it actually worked well enough. You put a vacuum in there and it sucks up dirt and it was a small enough constrained problem that it worked and it was successful and it actually cleaned people's floors and was good enough that people have been spending money on it. And, frankly, we kind of spawned an entire industry of vacuuming robots.

[00:06:17] JM: You work on the embedded systems side of things. How do you define the term embedded system?

[00:06:25] CS: Sure. That's a good question. And everyone has a slightly different definition, but I think of an embedded system as a single or a non-general purpose computing system. So it's any type of electronic system. Usually has a little microcontroller in there, microprocessor. Some little computer in there that's controlling it, and an embedded system is a non-general purpose one, which means it has a specific purpose or purposes in mind.

So Fitbit is a great example. Your digital watch or an old school mp3 player or even an iPod. Not an iPhone necessarily, but an iPod, where this thing is a music player or this thing is a pedometer or this thing is a digital watch or this thing is a Roomba. These are all embedded systems.

Once you get to a device that has like an iPhone, which basically if it has an app store, it's probably not an embedded system. Again, people can argue about the definition, but that's kind of where I draw the line, a non-general purpose computing system. Usually they're characterized by having pretty hard constraints in terms of power usage, in terms of batteries, in terms of memory sizes, in terms of computing power. They're small systems, which means you don't just cram a giant x86 processor in there with fans and memory and a wall plug and a liquid cooling system. Instead you have maybe an 8-bit microcontroller, maybe a 32-bit microcontroller. Maybe you only have you know 32k of code space and maybe 8k of RAM available to you, or maybe you have something running embedded Linux, which has a gig of RAM and significant computing resources. But it still can't run forever on a battery. So

Raspberry Pi is kind of a higher end or could be the brains of a higher end embedded system. That's kind of how I think about it. Does that make sense?

[00:08:12] JM: Absolutely. Tell me a little bit about the hardware-software interface for a robot like a vacuum robot.

[00:08:23] CS: Also, like any embedded system, essentially the embedded system team, the embedded software engineers, the embedded hardware engineers, the electrical engineers come up with a circuit board and basically the embedded system software is the stuff that needs to control any of the hardware that is in the product. So if you think about a Roomba or really any robot, the robot generally has to move and sense the world. And so I tend to think about the job of an embedded software layer or an embedded team, embedded software team, their job, our job is to make a device move and sense the world. So you've got motors on there that make the various brushes spin, make the various vacuums turn on and off, makes the wheels spin, and then you have whatever kind of sensors are on the robot. And our robots, depending on which robots you're talking about, have a variety of different sensors that let the robot figure out where it is. They figure out what it's interacting with. And even interior sensors to tell what is my battery state? What is the temperature of my various motors? What is the torque on my various motors? How much current my various motors are drawing? That kind of stuff. Does that answer your question?

[00:09:38] JM: Tell me about some of the modules and sub teams that you oversee as an engineering manager. What are the projects? Just to give people more of an idea of the kind of things that you that you work on.

[00:09:51] CS: Sure. I mean I can't give too many specifics, but as one would expect, as any company does, we have a variety of new projects, new robots, new types of robots, new incarnations of existing robots and some of them need new hardware and software. And some new projects are complete ground-up redesigns and some new projects are tweaks to existing projects. And this is not unique to the embedded systems or robotic space. This is any project anywhere, right? You have kind of a 1.0 to 1.2, kind of our 1.0 to 1.1 project, versus you have

a, “All right. Clean slate. We're doing a new system.” And so we get to manage projects. I get to manage projects and work on projects kind of across the board depending on when projects are in different phases of their lifetime.

As well, we have robots that can be updated in the field. And so we are fortunate that we're able to deliver updates to robots after customers have bought them and brought them home. So we can provide value to customers with new features, with that kind of stuff, once they actually have it in their houses. So we also get to work on robots that have already shipped and we can't change the hardware at that point, but we can upgrade the software. And so we get to work on a variety of projects like that. As well as at any company, you'd expect there's sort of a researchy kind of the house. And so we have different types of evaluations that we're always doing. Looking at new sensors or new different robot types or new different systems. Basically anything that we might be able to incorporate into a robot or a project or a product in years to come. Again, like any company, we do some of the kind of we're shipping this product soon and we do some of the looking ahead kind of advanced development work. And the people that I work with, we work on all that across the board.

We're a mid-sized company. We're about a thousand people or so, a little more than that maybe. We're not huge, but we're not small. And so it's a good size where we get to work on a broad base of pretty interesting projects.

[00:11:51] JM: How do you ensure the safety of a robot? I mean a robot is a thing in a physical environment. A vacuum cleaner sounds innocent, but I imagine there are things that a vacuum could do that are unsafe. So how do you ensure safety?

[00:12:07] CS: That's a big question, and in fact there are perhaps no surprise to you. There are international standards that govern the safety of home robots. There is a standards body called the IEC, and they have standards on most any types of electronics you might have in your home and outside of your home. And so we have to certify our products to various standards for all types of standards. There's not just one standard. There's a bunch that we have to meet and any consumer electronics product has to meet. And so there's safety –

Sorry. There's software and there's hardware sorts of checks and testing and assurances that we need to give both from the design. Like the system must be designed to work in a certain way and then the system must be tested to a certain way and the system must fail safe in a variety of conditions. And so that's kind of a high-level view of it, but yeah, that's a big part of the job at iRobot at any robotics company. I mean you start to think about robots in the manufacturing spaces and self-driving cars, and of course that's a whole another level of safety. But it's a similar end goal that the product should do what it's supposed to do and nobody should be able to get hurt through fault of the product or fault of the user.

[00:13:26] JM: What's your system of testing? How do you test robots?

[00:13:31] CS: Very carefully. We have all kinds of testing. I mean as any real world physical product that has to bump around every house in the world, we have a ton of in-home testing. And so our employees are testing stuff at home all the time. We run all kinds of simulator tests as well. So we have a pretty extensive simulator environment where we can run most of our software in a simulator in simulated homes and we can configure them in many, many different ways. We have a bunch of internal test labs as you might expect in our in our buildings to test different floor services and carpet surfaces and basically any kind of home surface you might consider.

In North America, where I think you and I both are, there's a lot of carpet. There's some hardwood and there's some tile. In a lot of the rest of the world, there're all kinds of other different types of floor surfaces. I had never heard of it, but I guess in some countries there's tatami mats, which are it's kind of a very thin sort of a paper-like floor surface, which I'd never heard of. But again, we have to make sure that we work well on those and every different floor type.

If you go to New England for instance, you're going to find a ton of basically thresholds, room to room thresholds that are an inch or so, up or down from one floor, from one room to another just because some of the houses are older, the architecture is different, that kind of a thing. And so we have to try to test in all those situations. So we use just running around people's

homes. We use a bunch of simulator testing. We have a bunch of automated testing that we do using a bunch of stuff that we've built ourselves as well as a bunch of stuff we've taken from elsewhere. And we do all manner of real and simulated testing.

[00:15:14] JM: Let's get into the software in a little bit more detail. One thing I wonder is if you're building a robot, do you think of it as just a single node system or do you need to think about distributed systems when building robots?

[00:15:30] CS: I'm not going to answer that for a robot. Basically I'm going to answer that for any embedded system. So if you have one microcontroller, sort of the computation in embedded systems is a microcontroller and you can call it a processor, you can call it a chip, you can call it microprocessor, whatever. But the microcontroller is the kind of canonical word. So if you have a one microcontroller project, if the functionality can be done with one chip, then that's sort of the single, the smallest unit of computation, then you don't have to think about any distributed anything. You have one chip. It's got a couple sensors hooked up to it. It reads the heart rate sensor and the step sensor once a second and you're – I'm sort of describing a Fitbit here just because it's something that most people are familiar with. And there's one chip in it. You don't have to worry about any distributed processing. You don't have to worry about talking to anybody else. So there you go.

Once you start talking to the cloud, which many projects do today. IoT is a buzzword, but it's also sort of everything most consumer electronics or maybe many consumer electronics devices have some internet capability. And so once you're able to talk to a cloud, a back-end server, or even just a phone via a Bluetooth connection or something, you start to have to think about, “Okay, I have perhaps poor network between me and someone else I'm talking to,” and then you start thinking about distributed systems.

Additionally, our robots don't just have one microprocessor in them or one microcontroller in them, they have several. And so then you start having to think from kind of that you have a functionally distributed system where one microcontroller is in charge of one set of functionality, another one is in charge of something else, and a third one is in charge of

something else and they all talk to each other over some sort of a protocol inside the robot and you have to design the system. So the protocols and the communications paths work well 99.99% of the time, but you know that physics gets in the way and ESD, electrostatic discharge, happens, random stuff happens which you try to minimize but you also have to plan for. And so you need to handle lossy communication just like in any sort of traditional web distributed system. So that was a long-winded answer. I'm not sure if I answered your question.

[00:17:52] JM: It sounds like the short answer is yes you do need to take distributed systems into account in a pretty regular basis.

[00:17:59] CS: Yes. Yes.

[00:18:00] JM: But let's talk more about the single node. So I believe that a robot uses a real-time operating system. How does a real-time operating system compare to a normal operating system?

[00:18:17] CS: Good question. So real-time – So let's step back. So forget about a robot. So a real-time operating system, or RTOS is the abbreviation that people use for that, a real-time operating system is an operating system that gives you a guarantee that things will happen in real-time. Now what real-time means is it's running on microsecond or perhaps even faster sorts of time frames. And so if you're running Linux at home, if you're running Windows at home, if you're running Mac at home, you as a programmer or any program can ask for some CPU time to do something. And you can get some rough guarantees that, okay, you'll probably be able to run at a reasonable interval, but you may not get to run for microseconds or milliseconds or even whole seconds. Your process, your program may not get to run for who knows how many seconds. It may not get to run on a real-time operating system.

The guarantee is that as long as you, the programmer, have done your job, the way you set it up is you set up some frequency and you say, “Okay, every four kilohertz, every four thousand times, four thousand times every second, I want to read this sensor and then I want to do

something with this data." The real-time operating system is an operating system that has very low overhead, very low footprint. It is not doing a ton of stuff other than switching tasks around so you can do task A and task B and task C. You can read sensor one, read sensor two, read sensor three. Process data from sensor one, process data from sensor two, process data from sensor three. And the RTOS is small. You can understand it and you can see deterministically that, "Okay, it's going to give me the response time I need. My application determines that I need to read a sensor 500 times a second or 10,000 times a second." And so you'll use a real-time operating system in order to manage the overhead of the chip and to get your work done.

Now, you actually don't need a real-time operating system on a chip. You don't need any operating system on a chip. The very simplest embedded systems, especially those that are just a single chip, won't use a real-time operating system and it's a system that we call bare metal, which means there is no OS real-time or otherwise, and instead it's just your program running directly on the chip. And so your program basically controls everything the chip is doing with no operating system to help it, with no task scheduler to help it, with no memory management software to help it. Instead your processor just reads a sensor processes data, spits out some output data.

So bare metal versus RTOS, that's one of the big decision points in determining what kind of a system or what kind of an architecture you're going to use on any given embedded system. And a given embedded system, if it has let's say five microcontrollers in a system, three of them might use RTOS and two of them might use no operating system, it'd be a bare metal. And nowadays embedded Linux chips or chips that can run Linux, which we kind of call in embedded world, there's an embedded Linux flavor, those can also be in a system. So you can have a system that might run an embedded Linux processor, which is not real-time. You might have one or two or three processors that can run an RTOS and you might have a couple processors that run no OS and they're just bare metal. And there's a ton of tradeoffs and there's a ton of it depends conversations you need to have to determine what the right balance is for any particular task or any particular microcontroller's task in that system.

[00:21:49] JM: Can you say what operating systems you use or which ones you can use?

[00:21:55] CS: Are you asking about iRobot specifically or –

[00:21:57] JM: Yes.

[00:21:58] CS: I'm actually not sure what we can say. I can tell you that we're using much like anybody who's written software who has products who've been around for a while. We have a couple of homegrown operating systems, RTOS's in our systems. We use some more commercially viable ones. As a company, I'm sure we've said things about what we use, but I'm not familiar enough to go out on a limb and say. So use Google and see if people who are actually cleared to speak about it have said something.

[00:22:24] JM: Gotcha. How do you debug robotics software?

[00:22:30] CS: Yeah. That's tough. It is an art. It is not a perfect science. Decomposing functionality down to the smallest sub-layer that you need to play with is the way all computer science and all computer engineering works. So when we architect systems, if you have – Let's say you have three microcontrollers in a system and one of them handles the motors, one of them handles the sensors and one of them handles the user interface. I'm just making this up. You try to design your system in such a way that you can observe just one of those processors doing its thing independent of the other two. So if it's a motor controller, for instance, you hopefully have some way to sort of tell that chip, "All right, run the motors at a certain speed," and then you can hopefully observe what it's doing.

And in the embedded systems world, we frequently use hardware debug tools like oscilloscopes, like logic analyzers, like multimeters that let you read voltage levels and a variety of more expensive equipment as well. But we're down in there with a schematic of a circuit, with the circuit board. Most of the robots that we work with are not as we call them buttoned up, but instead we've got the top off and we know we're able to probe at the circuit board to

see, “Is the problem in hardware? Is the problem in software?” as we're developing the functionality.

And so if you can design your system so that the different nodes or the different microprocessors are kind of independent and then you can get in there and hopefully you've left yourself enough hardware and software kind of breadcrumbs to figure out what in the world is going on in your system. Now that's at a chip level. There's various ways to debug stuff at a system level as well, but those are far more complicated.

[00:24:19] JM: What do you need to know about compilers to build good robotic software?

[00:24:26] CS: I suppose you don't need to know anything. I would say that to be a good embedded software engineer, and this is getting less true as chips are getting faster and more powerful for less power usage. So perhaps this is a statement that I might take back in five years, but I would say the more you understand compilers, the more you understand the code that you're writing, and most embedded code is in C your C++ nowadays. The more you understand what your compiler might do to your code, the more you can sort of guess as to how as it's going to perform.

When you're talking about reading sensor data 4,000 times a second or 10,000 times a second and then having to process many bytes of data and then do something with it 10,000 times a second, on a microcontroller that might just run at eight megahertz or 24 megahertz. You have to have a kind of a feeling of what the actual compiler is going to generate. You don't have to know, “Okay, there's an ad instruction and a multiply instruction and a load instruction and a store instruction.” But you have to know, “Is the C code or the C++ code that I'm writing, is it going to be expensive? Is this something that is going to be able to be done in less than a microsecond? Or do I need to basically – This is going to take seconds to compute this data and I can't expect 10,000 times a second updates on a particular piece of data?”

So knowing what your compiler is doing, knowing what your OS is doing, knowing what your real-time operating system is doing as far as scheduling a particular task becomes –

Nowadays, I'll say it's very helpful. Bordering on, absolutely required. That's more of a philosophical conversation. But yeah, having a background in compilers and operating systems in electrical engineering is all helpful to understand what's going to happen when you actually load the code onto an embedded processor and then let it run.

[00:26:19] JM: Do you use much static analysis?

[00:26:22] CS: Yeah. Yeah. Yeah. Absolutely. Absolutely. We and every other company that does this kind of stuff has a variety of different static analysis tools that run at compile time. Then you have plenty of offline tools that run. You have live profiling tools that run. It's not quite as mature. The embedded space doesn't have as good of tool chains, I would say, or at least not tool chains. The embedded space doesn't seem to have as much innovation in the developer environment as I see my friends in the web world have. But there's GCC and there's Clang and they're getting pretty good static analysis tools built into them nowadays as well as plenty of third-party, third-party static analysis.

[00:27:08] JM: Tell me about a deployment for a robotics company. Like you've got the hardware. It's already sitting at the customer's home. Can you just do continuous delivery to a robot?

[00:27:25] CS: So we don't do continuous delivery. We're not changing stuff every day. Like the physics of it are that we could, but there's a ton of reasons that we haven't gone down that route. I'm not sure what I can discuss, but if you think about any old internet of things system, let's pick on the Fitbit here, because I've never worked there. If you think about how often you push bits down to Fitbit, you're pushing that through someone's cloud, which means you're paying money for that. So that's one concern. You're pushing code down to millions of devices in people's homes and you can't control exactly where those devices have been and what state they're in. And so this isn't like a web server that you control.

And so how often you make changes is partially based on how much regression testing you've done. How confident you are that you're not going to brick someone's device. We use the

phrase brick a device to mean you load code to something or something happens on a device and you've turned someone's lovely new Fitbit into a brick, into a useless piece of hardware that has to go in the trash or get returned. So sort of the fear of bricking is very real and very on the top of mind as any internet of things type of a company thinks about deploying new software versions to any of their products.

[00:28:44] JM: What programming languages do you use?

[00:28:47] CS: Mostly C, C++ for the actual code that runs on the robot. And again, we're pretty typical of any robotics and embedded space. Most embedded companies, most embedded systems use C, C++. We use a lot of Python internally to do scripting and to do a lot of other stuff. It's kind of our glue language. And of course there's smattering over languages. Our older robots, which we're still developing actually use a dialect of Lisp. We are a 30-year-old MIT, more 30-year-old MIT spin out after all. So of course List has to show up somewhere in there.

So our initial robots, the initial Roomba and some of the ones that we're still making use a dialect of Lisp that we developed in-house to actually do most of the robotic control. But most of everything we do now is C++.

[00:29:35] JM: And if you were building a robot from scratch today, do you think you would use the same language or do you think perhaps Rust or Go or something would make more sense?

[00:29:46] CS: I would use C++ any day of the week. I mean Go, I don't know of it having a very good toehold and sort of footprint in the embedded space. I've seen people do it, but I've never looked into it very much. Rust has a very active and enthusiastic community around the embedded space. I see no reason to reach for it. C++, C have been getting – C++ in particular has been getting a lot better with C++ 11, 14, 17 and 20. And so I think it is still the language of choice.

Rust's evangelism is powerful and perhaps it will live up to its hype at some point. I haven't looked at it enough to render much of a judgment on it other than the simple economics of C++, you can hire a ton of people to do C++. There aren't that many people who do Rust. And sort of C++ and any known language is the language you know. And so myself, I've been doing C++ for 20 plus years, so you kind of know where the warts are and I don't know that the community understands where the warts in Rust are yet. And when I hear people say there aren't any warts. Then I know that they haven't used the system enough yet. So I guess check back with me in a year or two or three and we'll see where that goes. I'm sure that any Rust fans out there are probably yelling at podcast right now, but that's okay.

[00:31:09] JM: Tell me more about some of the canonical challenges of building software and how those turn up in your everyday work.

[00:31:18] CS: When you say canonical challenges of building software, could you be a little more specific?

[00:31:23] CS: Sure, for robots. I mean what are the things that just come up over and over and over again in your work and how do you prevent against them or how do you build practices or software libraries that make them easier to deal with?

[00:31:37] CS: Sure. Anytime you deal with real-world physical interaction and sensors, you're going to have – Sensors are imperfect and we're trying to sell a product that the average person can afford. And so we're not using a ten thousand dollar sensor on any of our robots, because then no one could afford it. And so when you're using low-cost sensors that we try to make them behave very well, you have to calibrate them and you have to understand what the range of acceptable values are. You have to figure out how a sensor or how a motor, how anything in the physical world will change over time, right? As devices age, as batteries age, their characteristics change. As motors age, their characteristics change. As sensors age, their characteristics change. And you don't think about that when you're using any of your electronics at home. But LEDs do actually wear out over time, which sounds insane, but they kind of do. And there're all types of aging that goes on.

So any system we do, there always is this sort of calibration and sort of a system wear element to any part of the system that we do. This isn't the purified world of software that runs on a server, and as long as the server has electricity you're good to go. It's a little more daunting than that.

[00:33:02] JM: What is unique about management of a software robotics team?

[00:33:07] CS: Good question. The cross-disciplinary nature of what we do is fascinating for me. Our projects and our robots and our systems, and again, this is zooming out again as I keep doing. Any sort of embedded systems project, any project that uses the real-world, the system is not just a couple of software engineers or even 100 software engineers. Instead it's a bunch of electrical engineers and a bunch of mechanical engineers and a bunch of systems engineers and a bunch of software engineers and it's not just one type of software engineer. It's a kind of a low low-level embedded software engineer. It's someone who does robot behaviors. It's someone who does the iOS or the android app and the cloud site app.

And so one of the unique challenges at least in the work that I've done as a manager and as an individual contributor is that you really have to work full stack. And when I say full stack, I actually mean full stack. Like down to the layer of I have to worry about what happens with electrons. What straight capacitance does to the sensor reading that my team is going to read into the microcontroller, that will then control some robot behavior, that will then send up to the iOS app, that will send to the cloud.

And so one of the most challenging parts of managing any kind of a sort of cross-disciplinary project is to get the communication flows going between different disciplines. Mechanical engineers spend their entire careers doing very different things than software engineers, and same with electrical engineers, and same with all these different types of engineers. I'm an electrical engineer and I've been doing computer engineering and software engineering for my entire career. So I speak a little bit of EE as we say, a little bit of electrical engineering, and I speak mostly software. And so getting the translations right and making sure everyone's in the

same page even though we have different disciplines is one of the unique challenges. And, frankly, it's one of the things I love about managing a team, leading a team, is making sure all the very smart, very technically capable people are all moving in the same direction at the same time on a project.

[00:35:10] JM: Can you tell me a little bit more about the software stack for a Roomba? I mean I guess I don't know if this is something you can actually talk about. Maybe it's too private, but I'd just love to know like how the software is even laid out or like the modules are laid out in the operating system and if you could talk about software architecture at all.

[00:35:32] CS: Sure. Sure. I can't talk any specifics about what we're doing, but let me try to give you the flavor for just a random off-the-shelf. Again, let's use Fitbit. Let's use a pedometer because it's kind of a small enough easy example. Fitbit's a complicated device and like they've made a great product. Don't get me wrong. I'm not trying to downplay it's product architecture. I have a lot of friends who've done a lot of work on the Fitbit. So if you all are out there, you've done great work. I'm going to try to simplify what you've done. But the software architecture, there's not a lot of different thinking apart from what any of you have done any type of software engineering have done. You want modules. I'm going to use the word modules, where a module is something that is self-contained that has some known inputs, known outputs, and hopefully it doesn't have to worry about who it's talking to.

You want to have some sort of a system architecture that is hopefully composed of a bunch of modules that if you want you could swap in and out when you need to and that you can debug and test or you can test and debug individually. And hopefully when you combine them together they all work interchangeably. And if you find a problem, hopefully you can figure out where that problem is. Fix that problem locally and not disturb the whole system and change the risk the way the system works. If you'd like, I can give kind of more of a concrete example of how that might work in something like a pedometer.

[00:36:51] JM: Sure. Please.

[00:36:53] CS: Again, let's assume we have a pedometer and it has a barometer altimeter for measuring height altitude and then some sort of an accelerometer to measure steps. So you'll have some sort of a main UI thread that's a little – Let's pick in real-time operating. So let's say we decide all right we're going to display data on some sort of little screen, a little LCD screen. We're going to need to take in readings from a sensor or two. So there're a couple more tasks. So we have a UI task. We have a sensor reading task or two. We're going to need to monitor our battery and do charging. So that's another very common task. We're going to need to probably do a bunch of math on the accelerometer data, on the sensor data coming in, because there isn't a sensor that says tell me when I have stepped. You usually have to do some floating point math and some wicked math on sensor data in order to get something that looks useful to a human. Like tell me how many steps I just took.

So right there, I forget how many I counted out, but that's five or six different tasks that you can expect your Fitbit type device to do. So at that point you're thinking, "All right, I probably want to reach for a real-time operating system." So you get a real-time operating system and you split the work that your system has to do into five or six tasks. And each task starts with just basically a C++ function that is kind of that task. And then you tell the real-time operating system, "Okay, task one is my UI task. Task two is my sensor, read the accelerometer task. Test three is my read the barometer altimeter task, etc."

The tasks should do only exactly what they need to do. So because we're dealing with hardware – Let's look at the accelerometer. So the accelerometer is going to be probably a little chip that sits off to the side of your microcontroller chip. And so the code, the module that reads accelerometer data, you want it to be modular. So you'd like to be able to swap out a different accelerometer chip in the future if you want to. So you want your accelerometer data or your accelerometer functionality to say, "Hey, I'm going to go read my acceleration, read the data from the sensor, but hopefully I can do it in a way that doesn't only work for this one particular accelerometer chip that I happen to have bought this week."

Instead you want to be able to change it out. So you will write an API that says, "Get accelerometer data." And then that will break down into something that says perhaps, "There's

three flavors of accelerometers we want to support, and so I'm going to – My flavor A, my flavor B or may flavor C," and each of those is a module, is a function, or a set of functions that can speak over physical wires to the particular sensor that you might have on your device.

And you repeat that process for the other modules in your system, for the other sensor, and then you go over to the data processing where you get the accelerometer data and you have to figure out where are steps in this noise of accelerometer data and you don't want it carrying where the accelerometer data came from and you don't want it carrying how big of an LCD you have on this little device. Instead you just want it doing some APIs worth of data crunching and then spit out a number every time a step is recorded or something like that, some known API.

Again, none of this is very different from traditional, any other type of software architecture, but you know there's hardware involved and there's just a slightly different vocabulary involved. Slightly different concerns than doing some random JavaScript program or some random any other type of program.

[00:40:50] JM: You've given lots of examples of how robotics software is a lot different than other kinds of software development, but maybe we can go through more, because most of the shows that we cover are interviews with SaaS companies, and that's very different than selling a physical product and having to do software development on a physical product. Do you have any more examples of maybe counter-intuitive things of how software development at a robotics company is different than a SaaS company?

[00:41:24] CS: I mean full disclosure, I have never worked on SaaS. I have never written a web program except for like a PHP site I stood up in like 2003. So I definitely don't know that side of the house at all. But differences between any kind of a physical systems product, the software architecture is still something you talk about. System architecture, decomposing system, system planning, that's all pretty similar. Upgrade times, again, we can't just deploy the site and have everyone using the same software instantly. There's the physics of time. There's the physics of bits getting across a wire. There's the physics of if your robot isn't online, you're not getting an update, that kind of a thing. So that is different.

If you mess up some types of code, especially if you mess up electrical design or electrical engineering design and the software-hardware kind of works but it works like 95% of the time but then you figure out, “Oh, there's a hardware problem with it.” You can't usually fix that with software.

Now the joke is that embedded software engineer's job is to fix hardware bugs, and it's kind of a joke and it's kind of true too where once a product has gone out into people's homes, again, regardless of whether it's a robot or a Fitbit or whatever, you can maybe make software updates to it, but you can't make hardware updates to it. And so we have to do a lot a very careful review of the hardware and the software and make sure they're playing well together before we ship it. Because if you've got a hardware problem, you ain't fixing that without some dramatic intervention. So that's kind of one of the concerns.

One of the interesting things that's come up in the last 10 years maybe, maybe five years, is I'll use the fancy phrase distributed computation. So let's say you have a Fitbit with Wi-Fi connectivity. Your Fitbit is small. It's got a tiny battery in it. It doesn't have a ton of horsepower, but it's connected to the internet. And the display on your smartphone is much better than the display on any Fitbit. No offense Fitbit people, but the iPhone's going to have a nicer display than the Fitbit. So you can do different display. You can do different processing on a phone basically off device rather than on device.

And so thinking about where should I do my processing? Where should I do my data storage on my device? Which is a sort of one-time cost that I have to pay for, and then customers have to pay for. Or do I do it in a website? Do I do it on the phone? Because more and more devices are connected to the internet nowadays, and network connectivity in some parts of the world is pretty good. We can start making interesting tradeoffs where data is processed. So that's kind of interesting. I'm sure there's an analog for the SaaS world, but I don't have a direct analog. So I know something unique we get to worry about.

In addition, simply the fact that our products are on every continent. They're in zillions of people's homes. And again, to make the Fitbit example, it's on zillions of people's wrists. Some people sweat more than others. And I had a Fitbit that got recalled because it turns out there was some nickel in it and it turns out I have a nickel allergy, I guess, which I found out only after getting this device.

[00:44:45] JM: Oops!

[00:44:46] CS: It was fine. Actually Fitbit handled it very well. I was impressed with the company the way they handled it. But you have to think about what metals and what plastics and what chemicals are in this product that you're selling and how will an eight-year-old kid use it? How will a two-year-old kid use it?

Roomba is a – Whatever. If you've ever seen anything on the internet involving a Roomba, it probably involved a cat sitting on a Roomba. And how do you deal with a 10-pound cat sitting on your product? Like do you test for that? Do you have to worry about that? If there's a camera on the robot and you have a cat sitting on it in the way of the camera, what do you do with that? So it's all the real-world stuff that we have to think about that makes it interesting. It makes it hard sometimes, but it also makes it interesting.

[00:45:33] JM: Let's think a little bit about the future. I'd like to zoom out a bit. I don't know if you saw this Amazon product announcement. There was basically a flying security camera that flies around your house, and if there's like an intruder, the security camera just like floats in front of the intruder and sort of videotapes him as he's intruding into your house. But I thought this was cool, because it was the first consumer drone that flies around in somebody's house. I've been waiting for – I mean the thing that I always think about, what I really want out of a consumer drone, is I basically want my smartphone to be replaced with a drone that like flies alongside me as I'm walking around outside and basically serves as my flying cellphone. I don't know how far we are away from that, but I just love to get your speculation on consumer robotics. Where we're going? How far it's going to – How long it's going to take to get beyond vacuum cleaners and mopping systems as powerful and impressive as those are?

[00:46:41] CS: Yeah. Interesting, I have seen that Amazon product. I guess I want to ask you a question. Actually so you said you kind of want to replace your smartphone with a drone that flies around next to you. Why do you want that? And I guess I'm going to challenge that a little bit. What do you want that to do for you?

[00:46:58] JM: Well, do you have a Google Home?

[00:47:01] CS: No.

[00:47:02] JM: Okay. Google Home is a great product. You walk alongside it and it does home automation. It does useful home automation for you. You can control your thermostat and can play your music for you. And if you had that as a flying drone thing, I mean the really long range thing I can imagine is you have this drone that flies alongside you. You could say, “Hey, drone, go get me a glass of water,” and it goes and gets you a glass of water and it comes back to you and gives you the glass of water and you can drink the glass of water. Or you can say, “Hey, play catch with me,” and you can throw a ball and it catches the ball and brings the ball back to you. Or you could say, “Hey, make a phone call,” and it'll like fly up next to your ear and do a phone call for you. I just want a really flexible like flying cellphone basically.

[00:47:54] CS: Yeah. So I guess if I can kind of – I don't know, talk about that a little bit. The idea of a personal drone is just kind of cool. Like we all like flying stuff, right? But the thing you want is a glass of water, or you want your lights to turn on and off automatically, or you want to play the song that you want to play, or you want that thing. You don't actually – I mean the drone is cool. So I was about to say you don't want the drone, but the fact is many people want drones. But one of the things I like about what our founder and CEO, one of our co-founders and currently our CEO, Colin Angle says, and I don't mean to be too much of a commercial here, but I like working at iRobot. I think it's a good company. Is that there's a job to be done basically. And the reason that Roomba have worked well and that our company is pretty successful is because we do something in the home that is useful for people.

So we clean, and we clean well. And so the next stuff to be done, whether it's at iRobot or any other company, is basically what is the thing that we want to help people accomplish in the home? And then what is the best way to do that? And maybe that's with a drone or maybe that's with another Roomba type device or maybe that's with some combination of your smart home, Google Home, Alexa, Apple ecosystem plus a robot, plus some other new piece of smart home equipment that we haven't figured out yet, plus Apple Airpods. I don't know. But I like to step back and say, “What's the thing you actually wanted? You want a glass of water? Okay, let's figure out what's a way to get a glass of water.Or you want someone to play catch with when you don't have anyone there with you. Maybe I'm being too picky on your initial problem statement, but that's kind of comes to mind for me.

[00:49:42] JM: Gotcha. Well, any other predictions for the future or things you're excited about you want to discuss?

[00:49:49] CS: I'm excited about actually being able to take vacations sometime outside of the home. I don't know. This is being recorded in early 2021. If anyone listens to this in the future, so the pandemic is still with us. I think that robotics, I think embedded systems is an exciting space. We have only barely scratched the surface of what is possible in terms of doing useful things for people that makes easier, it makes more comfortable, it makes more enjoyable to be at home to keep at home, to live in your home. We have a bunch of smart home stuff. When I say we, I mean society, has a bunch of smart home stuff and a bunch of fancy consumer electronics. In the next five to ten years or something like that I think we're going to see hopefully better quality devices. Things that actually help us do real things and hopefully some of the hype and the sort of the practical stuff, the practical sort of help me at home and help help me live a better life at home and help me sort of prioritize my family and the things I want to do win out over just kind of whiz-bang high-tech gadgets that do seem cool and do fun stuff, but do I really need a red LED glowing? Whatever. I’m making bad analogies for home products that I don’t want. But just to do useful stuff to help me enjoy my life basically. And I think the computing processing power and power requirements of it and cost of stuff is going – While the computing abilities are going up and the costs are going down and the power usage

is going down, so more is possible. So I'm excited about that and I'm hoping that we and everybody can do stuff to basically make everyone lives a little bit better.

[00:51:31] JM: Okay, Chris. Thanks for coming on the show.

[00:51:33] CS: Thank you very much. It's good talking to you, Jeff.

[END]