[Music]

Paul Cutler
Welcome to the Circuit Python Show. I'm your host, Paul Cutler. This episode I'm joined by Dan Halbert. Dan has over 40 years of programming experience with expertise in embedded software, databases, user interface, speech recognition, and more. Dan holds a PhD in computer science from the University of California Berkeley. Dan is sponsored by Adafruit to work on Circuit Python.

Dan Halbert
Dan, welcome to the show. Hi Paul, thanks Thanks for inviting me.

Paul Cutler
– I'm glad you're here. How'd you first get started with computers and electronics?

Dan Halbert
– So when I was in eighth grade, we had to do a math project of our own doing. And I had this idea, I was kind of interested in electronics and I subscribed to popular electronics. This is 1970, okay? And they had this kit from Southwest Technical Products Corporation, which made a lot of interesting kits, some music stuff and other things. And it was a, what's called, they call it a digital logic micro lab, which was this basically breadboard-y thing where they had like four NOR gates and some inverters and some flip-flops and a way of jumpering them together. And I was like, oh, this is great. I can like build this kit and then demonstrate digital logic. Ordered it. And then they wrote that it was gonna be backordered. Too late for me to get it for the math project deadline. My parents are physicists and they worked at Oak Ridge National Lab in Oak Ridge, Tennessee. That's where I was born and grew up. And my father said, "Well, you could learn to program instead, like, you know, you can use a computer where you worked." So I said, "Okay." So I like wrote some programs in Fortran, which were on punch cards, and had a good time doing that too. And I was still interested in the electronics. I was kind of had an interest in like computer architecture and programming. But it was interesting that that was kind of a critical like inflection point. It might've been like more electrical engineering than software, but for the fact that this thing was backordered. Later we went for a year to Belport, New York where my parents worked at Brookhaven National Lab. The high school there had just started to get a timesharing connection to a PDP-10 that was shared by a bunch of people in the county, bunch of schools. And because they were renovating the high school, we went on a split session. So I ended up having four hours in the afternoon every day with a bunch of other people to just use this time-sharing system. So I learned a lot about that, and I read a lot of books. And that kind of launched me. And I did a lot of programming after that, and went to college, and then grad school in computer science. And the rest is what has happened

since then.

Paul Cutler
– Well, starting in grad school and all the way up to today, how have you worked to make computing easier for people over the course of your career?

Dan Halbert
– That was a thing that I became really interested in when I was working on my bachelor's thesis. I read kind of a seminal thesis in that area. It was a system called Pyramelian by a guy named Dave Smith, who was at Stanford, and he worked with Alan Kay. It was just a really interesting, it was about a thing called programming by example, and it was also kind of using icons to program with. And I thought that was really neat. That's not what I did my thesis on, but I thought this is really neat and this is a way of making computers easier to use instead of having to teach people how to use Fortran or whatever. That's what I ended up doing for my graduate work. I worked on this thing called, this stuff called programming by example, which was like demonstrating a program as it operated. It would be kind of like using a programmable calculator, except that we would actually do the operations as you perform them, and then you could go back and substitute different numbers. That was kind of the basic idea. And it was done in the context of the Xerox Star System, which was the first system with a desktop metaphor, you know, icons and folders and things like that. And so I did it in a simulation of the Xerox Star System that I wrote in Smalltalk. And I thought that was really neat, but it was before user interfaces became really a thing in computer science. And so there were no jobs available in that field, really. And so I ended up doing a lot more stuff about programming languages because of my experience with Smalltalk, which was an early object-oriented language, and also one that had its own IDE, and it was sort of like its whole world. I didn't work on that for a long time. And then when I ended up taking the Adafruit job, which was not quite six years ago, it was like, here's this opportunity work on something that is CircuitPython, which is the whole point of it is to make computers easier to use. And so it's really a circling back of my original interests, you know, what I had been interested in long ago in the 1980s, and didn't come to fruition. And now it really has. And I'm

Paul Cutler
really pleased with that. That's fantastic. How did you first discover CircuitPython?

Dan Halbert
I had been paying attention to Adafruit's product line, but kind of only in a vague way, kind of like, "Oh, this is an alternative to Arduino stuff." And I didn't even have an Arduino. I was just just kind of, you know, armchair checking into this. We have a child with special needs, and I was on a mailing list of parents of people with

special needs, and I ended up getting in touch with somebody who had an adult son who was a filmmaker but who had cerebral palsy. He had trouble. He couldn't use his hands very well at all, but he had a foot pedal and a chin-operated joystick, which he used. He actually edited videos with this. It was quite a feat, and it's kind of slow and tedious. He had some troubles with the joystick. This His chin-out-for-head joystick was basically a game joystick that somebody had removed the handle of and stuck a screwdriver in it. So they would wear out after a while, and his father was saying, "Can you help us with this?" I didn't really solve that problem. It was kind of a mechanical engineering problem that he didn't solve, but I talked to him about what might make his life easier. What he really wanted was some way of invoking the keyboard shortcuts in a professional video editor that had a lot of drop-down menus and keyboard shortcuts. And I said, well, I could make you a thing with multiple foot buttons that would do this, and I could make it so that it was configurable, because that seemed to me to be important, because if I gave him this thing, I wouldn't want him to be trapped in not being able to say, I want to add this other keyboard shortcut. And so I had just been reading about CircuitPython was just getting started, and I read about what Scott was doing and saw the videos and stuff and saw that it presented a USB drive. And so I said, "Well, I can just have this guy edit a configuration file, you know, and have a circuit Python program that reads that configuration file." And there was no way to do this with Arduino or anything. So I built such a thing. It took a long time because there were various mechanical aspects. And also it didn't really fit on like a feather expression. In zero, there wasn't enough RAM and getting the display to work and so forth. But I ended up fixing a bunch of things having to do with mouse and keyboard support in CircuitPython really early on, and it was a huge amount of fun. I submitted pull requests as a volunteer, and then I thought, well, maybe I could do this part-time because the regular my day job at the time was not that interesting. I was contracting for a large research lab, a defense lab. I arranged to be able to work part-time. I got permission to do that from my contracting company, who was fine with it. But the defense lab said, "No, you can't do this because we buy Adafruit things," even though what I was doing there had nothing to do with that. It was a perceived conflict of interest. Then I went back to Adafruit and said, "Could I work for you full-time instead of part-time?" They said, sure." And then I just made this transition, and this was around August 2017. So then I became the second core developer after Scott on CircuitPython and have continued with that since then.

Paul Cutler
>> What advice would you have for someone looking to contribute to the core?

Dan Halbert
>> I would say you may have an idea of some feature that you want to

add, and if so, check with us in the CircuitPython dev channel on Discord, or you may have some idea of something that you want to fix, in which case take a look at the open issues. And it takes a lot of reading code, kind of figuring out how things work internally. But the best way to figure it out is usually to read, pick something that seems sort of like what you're doing already. And there's a whole lot of boilerplate associated with like the busio.i2c class, for instance. There's the Python interface, and then there's how it's implemented on the back end. And usually when we add a new feature, we end up taking a bunch of files, copying them, gutting them, and putting new things in. And that's the way you get started is that you just use the existing code as a template. You don't have to start from scratch. And that's how you end up learning how it works.

Paul Cutler
How has CircuitPython brought the new hardware like the PicoW or the NXP/IMX family?

Dan Halbert
So the PicoW, it's really, I mean, that's really just a Pico with a coprocessor. So that was simple. It was really just, or it wasn't simple, but it was like the only thing we have to add is the coprocessor, is support for the Wi-Fi BLE coprocessor. So there was a library already existing that we used, and that's what Jeff Epler did. But for the NXP, what it really is, is like We look at what existing SDK or other package that the hardware manufacturer supplies and make some decisions about. Maybe there might be multiple choices, or it might not be a very good choice. Maybe we want to work at the register level, and we would like -- it's easier to use the SDK if it provides the right features. So we end up, again, taking an existing implementation that's kind of as close in terms of how we want the build structure to work, copying the guts of that. And then we would start with digital I/O and basically, can we get LEDs to blink? Can we start controlling the GPIO pins? Then proceed from there through the most necessary classes like the bus I/O classes. And usually there's some complication. For instance, the NXP has a very complicated memory architecture. It has fast RAM and slow RAM, and there isn't internal flash on the chip. It's external. It's hard to get all that stuff to work properly. So you have to do a lot of studying of that. In our case, Scott did a lot of that. I did a lot of the NRF52840 original port, and that's a very nice chip. Like, any pin can connect to anything, for instance, to any peripheral. So it was actually a pleasure to do, and it had a very nice low-level API, and it was probably one of the easiest to do compared with all the rest.

Paul Cutler
You coordinate and manage the release of new versions of CircuitPython. When do you know it's ready for a point release or versus a whole new version?

Dan Halbert
So we usually have some theme for a major version, and we'll say like, well, we want to have these one or two major things, like we want expressive Wi-Fi support or we want NXP or something, And so we would lean toward that. We would add features toward that. But for a point release, what we usually say is, what are some minor additions and fixes that are not just bug fixes? We make bug fix releases. That's the third digit, like 8.0, 0.3, or 0.4. And of course, then we release those as soon as it's clear that there is some serious issue that we need to fix. And for the point releases, we usually just say, I usually just come to the point where I say like, "Okay, I have a bunch of pull requests. "They're not a bunch of them pending. "And so I can kind of come to a stopping point. "I'm not waiting for some desirable feature or fix "that's imminent in the next two or three days. "And so let's go ahead and make a release." And I think our aspiration is to release early and often. but without a lot of incompatibility churn. Okay, so that, okay, this release is definitely better. A lot of people are very anxious that they're using a beta or an alpha release, but except in the very early stages, usually that release is still more stable than the actual stable release because it includes more bug fixes. Because we don't backport all the simple bug fixes necessarily because they have workarounds or we can just tell people to use the beta, for instance. So it really is just like, let's accumulate a bunch of things. And when it gets to be, like there are 60 or 70 pull requests in each release, in each minor release, and that could only be like three weeks worth of work. It's actually like, I have to go through all the pull requests and write one line summaries to put in the release notes. And the less of that I have to do, the easier it is to make a release. So I'd rather make a release more often than less so often.

Paul Cutler
Where do you see CircuitPython going in the next year or two?

Dan Halbert
I think we're going to see more support for the i.MX chips. The Metro M7 RT1011 is the first one of those. Those are interesting because they're very fast. The Metro M7 doesn't have a lot of RAM so much, but for instance, Jeff Epler did this synth I/O thing, which runs very well on such a fast processor. So I think we're going to see more of that. We're going to see BLE support added to the PicoW at some point, probably, and better BLE support in the Espressif line. Espressif has a nice, a very interesting risk chip coming out that's not Wi-Fi based, but would be presumably be below cost and yet fast. And that's going to to be a very interesting chip, but we don't know about that yet. And I also think we need to not just keep adding features. We have a backlog of things that need to be fixed. And so I'd like to see some sort of increase in the overall quality to happen in the short run. And then also, I'd like to see more work being done with async.io, with async with cooperative multitasking. And we have

somebody who's very interested in that and has made some major changes. We need to review and take advantage of, but we've kind of been held up by getting the eight releases out. And so I hope to see more of that. And also what that means is, very importantly, is writing up more examples and using Async I/O in simple projects and giving people simple templates for using Async I/O so that even beginners can follow those templates and use it successfully and say this is a style of programming which is actually pretty natural because async.io in the past has been very abstruse and this if but if you use it in a very structured way I think that people can make progress with it and can use it in ways that and make it actually easier than

Paul Cutler
writing one big while true loop for their program. I agree with you I just wrote my very first async program using one of the learn guides and it was eye opening actually how easy it really was.

Dan Halbert
Yeah, it's like, is that all to it? Like, really? You don't have to worry about all the issues that come up with, say, threading where you have dangerous shared access to things, where things change out from under you, right? Instead, you're saying, "Now I'm ready. Go ahead." Okay, right. So I'm glad that you found it easy. I'm really happy about that.

Paul Cutler
So out of all the chipsets that you just mentioned, if you were starting a new project, which board would you reach for?

Dan Halbert
It's kind of funny. I don't have a whole bunch of projects that I want to do. I usually can say, "Oh, yeah, I could do that." But when I'm testing something or when I just want something to work reliably, I often reach for one of the Atmel boards because I know that, or now they're microchip. I know that the implementation is very reliable. And I also find the NRF implementation very reliable. It's not idiosyncratic. So those are the ones that, if I just want to see, like, well, does this work, for instance, like if I'm just testing a sensor or writing a library or something like that, I will go for one of the more mature implementations, ports.

Paul Cutler
Dan, thanks so much for being on the show.

Dan Halbert
You're welcome. Thank you very much for having me.

Paul Cutler
Thank you for listening to the Circuit Python Show. For show notes and

transcripts, visit circuitpythonshow.com. You can also now follow the show on Mastodon. Just visit the homepage and click the follow button in the banner. Until next time, stay positive.

[Music]