Oral History of Brad Cox

Interviewed by:
Hsu, Hansen

Recorded August 2, 2016
Manassas, VA

CHM Reference number: X7863.2017

Hsu: This is Hansen Hsu, in here interviewing Dr. Brad Cox at his home in Manassas, Virginia. It is August 2, 2016. And what do you prefer to be called?

Cox: Brad.

Hsu: Brad? Okay. So we'll start by asking you some biographical questions. So where and when were you born?

Cox: I was born in Ft. Benning, Georgia toward the end of the Second World War. Then my parents moved to my father's birthplace in South Carolina. He was a dairy farmer, so I spent my youth milking cows, taking care of the farm, walking in the woods, and collecting snakes. That was my [main hobby]—I was known far and wide for the snake collection. <laughter>

Hsu: So your whole family is sort of Southern heritage?

Cox: Yeah, pretty much.

Hsu: Do you have any siblings?

Cox: I have one brother. He's still in that area.

Hsu: Did you have any hobbies growing up? Or mostly—

Cox: Well, the snake collection and walking in the woods. Behind the farm, there was a sizeable lake that my grandfather dug, I think, with the help of sharecroppers or slaves. I don't know for sure; that was [well] before my time. But rumor has it either slaves or sharecroppers dug it all with wheelbarrows and shovels. And that's where I spent all my time was walking around that with a .22 rifle.

Hsu: Do you know what your parents ethnicity was?

Cox: Not exactly. We always guessed it was Scotch/Irish, but it was just a guess.

Hsu: Was your family very strongly religious?

Cox: Oh, yeah. I was the exception. I never understood it. I never got the memo. It just—it never sunk in, ever.

Hsu: Do you remember what denomination they were?

Cox: It was Baptist at one time, and then Presbyterian.

Hsu: Okay. And any political views?

Cox: Nothing stands out. Probably Democrat, but I don't really know.

Hsu: What was your education like before high school?

Cox: The proverbial—well, it was actually three rooms. Three-room schoolhouse until the 9th grade, and then I skipped the ninth and went to a high school in the nearest town. And that was, well, it was just a typical high school. The other one was atypically small. It was out way back in the boonies.

Hsu: Oh, wow. So until high school you were in basically like a one-room school. Three-room school.

Cox: Three-room. I'm overstating. The first, second and third grades were in one room. One building with three rooms. Then there were some out buildings for the next few grades, and then there was the high school that was more typical. So I'm understating. It was more than one room, but it was a small, and class sizes were small, and you know, everybody went to school barefoot in the summer.

Hsu: Oh, wow.

Cox: Pretty typical South, or stereotypically South.

Hsu: And what was your high school experience like?

Cox: Oh, typical. Nothing really stands out as being very different. I was pretty good at it. <laughs>

Hsu: Were you at the top of the class?

Cox: Yeah, top two or three. There were a couple of women that I couldn't quite beat. <laughter>

Hsu: And what was your favorite subject?

Cox: Probably Science.

Hsu: Science. Science, Math?

Cox: Yeah. I eventually ended up Chemistry, but I think that came later. I think when I was in college, I got focused more on Chemistry, but in high school, you don't really specialize.

Hsu: Right, right, yeah. Well, let's move on to that, then. Says you went to college at Furman University?

Cox: Yeah.

Hsu: Where is that?

Cox: Furman University is in Greenville, South Carolina. It's one of these Bible Belt colleges. And by that time, I was in total rejection. I just couldn't stand the religious part of it. It was a pretty decent college, except for the religious parts of it. I eventually gravitated to Chemistry. Organic Chemistry was my favorite. And at that point, I had the idea I was going to be a doctor when I graduated. A medical doctor. I eventually wised up and chose a different path. <laughter> It's not pretty what's happened to medicine since then.

Hsu: Oh, I see. So what was it about religion that turned you off especially?

Cox: I wouldn't say it turned me off. It's just I never got it. Nobody ever [took the trouble to] explain why anybody would believe ~~this~~ [in it], when [otherwise] this is more obviously true. I just never understood it. I regard it now as a fault. I mean, probably the most important ~~thing~~ [factor] in all of human history is religion, and history shows you that, but I never understood.

Hsu: Because you were more into science per se?

Cox: I was most of all into what everybody tells you when you're growing up, is, "Think for yourself." And thinking for myself didn't go that way.

Hsu: Okay. And what was it about Organic Chemistry that particularly attracted you?

Cox: I was good at it. <laughter> And it—I just enjoyed it. It was—I got these summer scholarships to do research, so I could stay there in the summertime and do research, and [it was] very exciting.

Hsu: What made you decide to pursue medicine, at least at that time?

Cox: Well, at that time, there's so much you don't know when you're a kid. You know, you don't know what the future holds, or what it'd be like to work on X, and you've never really heard of Y. But medicine I'd heard of, and that seemed to be the next step. And I only got off of that when I went to graduate school.

Hsu: I see.

Cox: I didn't go to a medical school, but I chose a different path about that time.

Hsu: So did you start moving away from that when you were choosing graduate schools. Because you would have had to decide between grad school and med school already.

Cox: I don't recall it being an issue. I'd heard of the University of Chicago's role in the war, and I think that's why I chose it.

Hsu: What was the role? What was their role?

Cox: Well, its role in inventing the A-bomb.

Hsu: Oh, okay. And so you wanted to be—so it was because of that, or because that's very far from medicine.

Cox: Oh, yeah, it—medicine or science.

Hsu: Oh, okay. Just the general science side, so—

Cox: Yeah, I steered more toward the science side as time went on.

Hsu: Right, okay. Well, what was it like for you growing up during the Cold War?

Cox: Wondering what all the fuss was about. <laughter> It's another one of these things where I never got the memo. I never understood what the—why anybody cared if somebody was communist, or why anybody would care enough to call themselves a communist. It just seemed like a—something I didn't understand, didn't want to spend time thinking about. <laughter>

Hsu: Okay. So what was your first exposure to computers?

Cox: That was very late. The first exposure to anything like a computer was a hand-cranked calculator in the Chemistry Department at Furman. And I enjoyed that thing, and practiced a lot.

Hsu: So it was like a mechanical adding machine?

Cox: I don't remember if it was mechanical or electronic. It may have been mechanical, what may have been early electronics. And it was a huge thing. So maybe mechanical. And then when I got to University of Chicago, I don't know what it was that told me this, but I chose the Department of Physics, Quantum Mechanics, because rumor had it that they had the biggest computer budget on campus.

Hsu: Okay.

Cox: There it is. That was the reason. Again, you don't know enough to make intelligent choices, so I was fortunate that I chose one that I was good at. And that was the first time I got involved in computers was programming IBM 7094 programs for molecular orbital calculations on punch cards. <laughter>

Hsu: I read that your degree was in Mathematical Biology.

Cox: Yeah, that came a couple years later.

Hsu: Oh, so you were first doing a Master's degree in Physics?

Cox: No, the University of Chicago lingo for it was Research in a Related Department. So I was still in the Physics Department, formally, but in reality what I was spending my time doing was Mathematical Biology.

Hsu: Oh, okay. And what drew you to that field?

Cox: <sighs> Most interesting problem I could think of to work on at that time was how the brain worked.

Hsu: Ah!

Cox: And to my astonishment, ~~that field of neural networks~~, [I gravitated to an early form of neural networks,] which is really what I chose [as my research topic. Even built a 64-cell neuron simulator on a 12-bit PDP-8I computer. But I] didn't have a prayer [of success] back then. We [just] didn't know enough. Computers were too tiny. You know, a long list of things caused me to eventually get out of it, 'cause I knew it was going to be hopeless. But the astonishing thing is within the last five or so years, it's beginning to catch wind big time.

Hsu: Yeah. <laughs>

Cox: So I'm not sorry I left it, or it would have been a really miserable 20 years.

Hsu: <laughs>

Cox: Waiting for the breeze to come. But that's really what I started in.

Hsu: Yeah, I mean, your first paper was on Simulation of Neural Nets, correct?

Cox: It was—if you'd call it a paper. I wrote a little thing for DECUS, DEC Digital Computer User Group. Shows the computer I was doing my neural networks on, it was a PDP-8, with a teletype and an oscilloscope, no persistent screen and all. And I could draw dots on it fast enough to show its neurons were firing, and [it was] the most primitive thing you can possibly imagine.

Hsu: <laughs> What was your first post-doc?

Cox: Pro—?

Hsu: Post-doc. Did you do any post-doctoral pursuits after you graduated?

Cox: That neural network project was where I started on my post-doc. That was—I eventually—I realized very soon that that was hopeless. We just didn't have enough knowledge to make any progress on it. So I got involved in mathematical modeling of squid giant axon firings. And got a—that was my degree paper.

Hsu: Okay. And so that led directly to your post-doctoral work. Or that was your post-doctoral—

Cox: Well, it was my doctoral work, which led directly to post-doctoral at Woods Hole [on real squids.]

Hsu: What was your first job after post-docs?

Cox: After post-doc, I had a look around at what I enjoyed doing and what I was spending my time doing, and eventually just realized that what I enjoyed about the whole process was computer side more than the science side. So I took a couple of jobs—two jobs [in sequence,] working for newspapers that wanted to automate their newsrooms building these huge gold-plated newsroom automation systems. One for *Chicago Tribune*, and the other for the *Toronto Star*. And then from there I got into ITT and the more recent history.

Hsu: So that's a good segue. So how did you find yourself at ITT or International Telephone and Telegraph?

Cox: Yeah.

Hsu: Is that a—because I mean AT&T was the monopoly, right?

Cox: Yeah, but they're different companies. ITT is International. AT&T is American. But ITT at the time was International Telecommunications.

Hsu: Where is that headquartered? Or was headquartered?

Cox: I'm not exactly sure where the headquarters was. It's International, so much of its bulk was in Europe. It may have been there at the time. No, I think it was New York. Rand Araskog was the President, and I think it was around New York. Don't hold me closely to these facts. I'm a little tenuous on the structure of that company. But they were all busy inventing the use of object oriented programming for building telephone nets. Telephone switches, and built a big research operations for that. And that's where I met Tom Love and started down this trail.

Hsu: Right, so you had met him—well, he had recruited you into the company, and so you hadn't known him before that?

Cox: No.

Hsu: So why did he recruit you in particular?

Cox: You know, I don't remember. Geography had something to do with it. I was living in that town near where the company was, but I don't recall how we got connected.

Hsu: And where was that? Was it New York or Connecticut?

Cox: Connecticut.

Hsu: Connecticut. And you mentioned they were using object oriented programming for network applications. Were you on—

Cox: They didn't call it that, but it was—the way telephone networks are built today is with these self-contained computers basically. Before that it was all these mechanical rotary switches and all that. So ITT was building the first step away from the old way of doing telephone switches. And in retrospect, it behaved like an object-oriented system. They didn't think of it that way themselves at the time. But that's, in retrospect, that's—

Hsu: Because it was encapsulated? Is that why it was, in your mind it was object oriented?

Cox: Yeah, and in some way that would make—you'd have to talk to a telephone engineer to get the details on this. I was working in computer engineering, not telephone engineering. So, but a good friend of mine, Ken Hamer-Hodges was a lead designer of that system, and he's the one that made that claim later.

Hsu: Oh, I see. How—actually how familiar were you with object oriented programming?

Cox: Not at all.

Hsu: Not at all? I mean, I think somewhere you mentioned that you met Adele Goldberg in graduate school at Chicago?

Cox: Yeah, mm hm.

Hsu: I guess, maybe elaborate on that?

Cox: Yeah, when she—University of Chicago had a nascent Computer Science Department. She was part of that. And she, and several others, was sort of generally on my radar screen, but they're not particularly close. I knew who she was, and so there's not a lot to say about it.

Hsu: Did you have a lot of contact with the Computer Science Department in general in grad school?

Cox: Informally. Largely around the keypunch machines. There was not a lot of formal contact.

Hsu: But you were aware of the think, the ideas coming out of the department.

Cox: Not object oriented programming. I don't think she'd heard of it back then.

Hsu: Right, that's before she went to Xerox, probably, yeah.

Cox: Yeah. Mainly, remember everything was punch cards back then, and that was generally how you met people was around the punch card machines.

Hsu: Oh, okay.

Cox: So. <laughter>

Hsu: Waiting to give them your stack?

Cox: Yeah. <laughter>

Hsu: So what sorts of programming language were you familiar with when you got hired at ITT?

Cox: Assembler.

Hsu: Assembler, for which architecture?

Cox: The IBM 7094, IBM 360 and the PDP-8. All three were largely Assembler. And then I was mainly—I was campaigning to use Fortran for some of these molecular orbital calculations, but not successfully.

That—Fortran was too slow, you see? <laughter> So I knew Fortran, but I don't think I ever used it for serious work.

Hsu: Oh, okay.

Cox: So pretty basic.

Hsu: Oh, wow, yeah. So what did you do for Tom Love when you got hired at ITT?

Cox: This was a Research Department with a handful of people—with four or five people in it. I was one of the four or five. And one project was to use program generation approaches to buildings applications. The whole idea was, we would search for ways to make productivity improvements [that ITT might adopt.] So one theory was that program generation was—had the legs to make a difference.

Hsu: By program generation, you mean the—

Cox: Programs that write programs.

Hsu: Okay.

Cox: Then UNIX began to get onto my radar screen. By which I mean, if you could crowbar it out of AT&T's hands, it was a way to get at that, I mean, nobody was pushing it. But we thought it could make a difference to the productivity of the people building this telephone switch, especially for testing these [new] switches. So we found [this] company [Onyx] that was building this UNIX box about the size of that coffee table there with handles on it. And I built a script basically that would let you control a set of [debuggers—connected to] these computers that made up a telephone switch, so you could control more than one by using this script. And that made a huge fuss, and I ended up getting sent to Europe to introduce this thing to the people that—oh, gosh, I've forgotten the company. Philips. [For use by the people that were responsible for testing the switches.]

Hsu: Oh, Philips.

Cox: Yeah, Philips' big lab in Stuttgart. So I made a big impression on a whole bunch of people. And then I think about the third thing I got involved in was building the ancestor of Objective-C.

Hsu: Oh, this was at ITT?

Cox: Yeah.

Hsu: You started there?

Cox: Right.

Hsu: Before we get into that, so why was the group—so the group that you were in—I mean, you write in the History of Objective-C paper, you called it the Frame Group.

Cox: Oh, Jim—

Hsu: Jim Frame?

Cox: Jim Frame was the Head of it.

Hsu: Was the Head of it. He had been hired from IBM, originally?

Cox: Yeah, a lot of those folks came from IBM. He brought most of his acquaintances. It seems.

Hsu: And was that the reason why there was this focus on productivity?

Cox: Yeah, that was the banner of the Frame Group. And the mission of that group was to make a difference in productivity.

Hsu: Okay. And was that—was it at that time that you became familiar with Fred Brooks' work on software engineering? *The Mythical Man-Month?*

Cox: Fred Brooks. It's hard to say when I first heard of Brooks. It wasn't through the Frame organization that I recall.

Hsu: It may have been later?

Cox: It may have been later, it may have not. I just don't recall.

Hsu: Okay, I see. So you said that the very beginning of Objective-C started there. So how did that project begin? What led to that?

Cox: Oh, I'd been looking for—I wasn't happy with C as a productivity foundation. And was casting around for anything I could find that might help. And that was about the time the *Byte Magazine* article came out.

Hsu: The Smalltalk *Special Issue*?

Cox: Yeah, the Smalltalk.

Hsu: 1982, I believe.

Cox: Yeah. And there were a whole bunch of things that I had—that I vaguely thought might help. So encapsulation I was pretty sure about.

Hsu: I see.

Cox: That was a definite. But—

Hsu: Why is that, in particular?

Cox: Well, C is so bad at it.

Hsu: <laughs>

Cox: You know, there's no way to—there are no objects in C. Essentially everything is public. There are no—you know, it just turns into soup.

Hsu: Right.

Cox: And the pain from that was what I was trying to escape. So—but there were—remember this was—it's—looking back, it's easy to forget how little we really knew then. [Email was in its infancy.] There was no internet. There was no—anything we take for granted today that didn't exist yet. So there was some vague ideas I had where Smalltalk might be better than C. And one of those vague ideas was it might be more supportive of building graphical user interfaces than C was.

Hsu: Right.

Cox: In retrospect, that's just bogus, isn't it? It just doesn't help that much.

Hsu: Oh, really?

Cox: Yeah. It helps, but it—but objects help almost anything you do. It has no particular advantage over any other object-oriented tool for doing graphics.

Hsu: Oh, you mean Smalltalk in particular.

Cox: Right.

Hsu: Over any other object-oriented language.

Cox: Right.

Hsu: But at the time, there weren't that many. I mean, there was Smalltalk and Simula.

Cox: That's it! That's all there was! So you asked what I saw in that paper. Those were two ideas that I saw in the paper. And in retrospect, I think the second one, the graphics thing was not particularly profound, but the encapsulation was. And garbage collection.

Hsu: Oh, okay.

Cox: That also seemed important, but I couldn't see a way to do it in C without losing the reasons that made C worth having. So garbage collection was never—I spent a lot of time working on it and worrying about it. But never successfully.

Hsu: Right. Can you maybe explain, like, what's the nature of that incompatibility?

Cox: Why it's so hard in C?

Hsu: Yeah.

Cox: Well, C is a frame-based language. You know, every time you call a subroutine, you just—you know, it just adds a stack frame to the end of the stack. And there's no flexibility in that. So it's easy to add things on the stack and pop things off, but other than that, your only other option for storing anything, an object, if you will, is to put it on the heap, where there is no stack and there is no structure. Now you could—there's some steps along the way to getting garbage collection that are compatible with that. Reference counting being one of the first. But nothing comparable to a Java-style environment where everything is—essentially everything is on the heap. And very tightly integrated into the language itself. C is tiny in comparison with that, and it's just very hard to do well.

Hsu: Right, now you mentioned that you wanted to preserve that aspect of C. Why was—

Cox: Well, that was the—I didn't want to do it poorly in C. I didn't want—I don't know. I just struggled to do it well, and never managed.

Hsu: Right, okay. Stepping back a little bit, I read that you and—was it Alan Watt, who was a classmate—oh, Alan Watt was working with the Frame Group at the time?

Cox: Yep. [He was a co-worker, not a classmate.]

Hsu: You and Alan Watt visited Bill Joy, who was Watt's classmate from Reed College. What was that meeting like?

Cox: Ooh, that was a long time ago! <laughs> But I remember it well.

Hsu: What year was that?

Cox: I have such a hard time with years. I think in the '80s sometime.

Hsu: Was it before the *Byte* magazine, the Smalltalk *Byte Magazine* came out?

Cox: It would have been after, because this was at—no. Probably after. This was at ITT that we went out there. For the first visit out there, we visited Sun in its garage, the proverbial garage. And then a few years later, I went out with Alan Watt to see Bill Joy. He was in his graduate student office.

Hsu: At Berkeley?

Cox: Huh?

Hsu: He—

Cox: [Yes.] At Berkeley.

Hsu: Okay. He hadn't joined Sun yet.

Cox: I don't think so. He may have [held a dual appointment by then.] I don't remember.

Hsu: Okay.

Cox: But if you ever met the guy, he talks a mile a minute. I mean, just machine gun fast. And I'm trying to remember now why we were there [why Alan wanted to visit him]. I think [that was while] we were trying to choose computers for ITT. And trying to get support for doing it using UNIX instead of one of these proprietary operating systems [that were] around. Don't remember exactly. But it's true. Met Bill Joy in his office.

Hsu: <laughs> And did anything come out of that meeting?

Cox: <clears throat> I don't remember anything tangible. We did wind up using Sun computers. Maybe that was the outcome of it.

Hsu: Okay. What was it about C and about UNIX that you guys were so committed to?

Cox: There was no other option in those days, except proprietary ones.

Hsu: Oh, I see.

Cox: And—

Hsu: But why was it so important to not choose a proprietary option?

Cox: <heavy sigh> 'Cause you—we didn't want a company the size of ITT to be locked into some other particular company, probably DEC.

Hsu: Mm-hm.

Cox: I mean in retrospect, we—we were on the right side of that war.

Hsu: Right.

Cox: But it wasn't at all clear at the time, and so we were looking for whatever—to mobilize whatever support we could get, thus Bill Joy.

Hsu: Oh?

Cox: And—I had just become incredibly annoyed with proprietary languages that—like Fortran, for example. Where—

Hsu: Oh, Fortran was IBM—?

Cox: Well, everybody had offered Fortran, but everybody added their [own] particular [herbs and] spices to it, [hoping to lock you in.]

Hsu: Ah.

Cox: So if you ever touch that—those spicy parts, which were always the attractive parts, too—

Hsu: <laughs>

Cox: —then you were—you're stuck and couldn't get out of it—

Hsu: Right.

Cox: —in the future. And that was just not acceptable.

Hsu: Okay. I mean, you mentioned the other languages like Cobol, Ada, Pascal, didn't have the right qualities either?

Cox: Well, Ada, that was about the time of government's full court press, and—our objection to that was, there's nothing in there that's demonstrably better than what you get for free with C.

Hsu: Okay.

Cox: So Pascal, that—that turned up later to be the main competitor at Apple.

Hsu: Mm-hm.

Cox: But it wasn't widely available. You—you couldn't use—I mean C was just, seemed like the obvious choice.

Hsu: I see. Because it came with every UNIX machine?

Cox: Right.

Hsu: Right. So going back to the Objective-C, so the—so this is about the time that you created the first Object-Oriented Pre-Compiler that was the, sort of layering Smalltalk on top of—on top of C?

Cox: Right. That was at ITT in Tom's group.

Hsu: Okay. And for you, how did—how did you see that in addressing the goals of improving productivity and coordination?

Cox: Well, it—it addressed productivity by bringing encapsulation to a language that didn't have any.

Hsu: Mm-hm.

Cox: And basically, the way we were thinking of our role there is trying to get object-oriented programming out of the research labs. Nobody had heard of it by then, all right, except for two or three research labs, mainly Xerox PARC—

Hsu: Mm-hm.

Cox: —for most of those. So we're trying to get object-oriented programming out of the research lab, and so we had to embed it in something that we could find on the factory floor, like C.

Hsu: Mm-hm.

Cox: Something that would be acceptable there. Smalltalk wouldn't.

Hsu: Right.

Cox: So—

Hsu: And C was already in wide use in the industry?

Cox: Wide—I wouldn't call it wide, but it was well-known and becoming established—

Hsu: Okay.

Cox: —by then. There were even companies building these suitcase-size UNIX machines by then.

Hsu: Mm-hm.

Cox: So it was—so it was [a] coming environment, but not yet fully. Not what it is today.

Hsu: Right. Where did the keyword ‘id’ for an object reference of any type come from?

Cox: Oh, almost every—yeah, ‘id’ word was coined like almost everything else in Objective-C that's not in C. We just had to coin terms and it was the two-letter acronym that—that was easy to use and an easier name than Pointer.

Hsu: Oh, okay. <laughs> So does it actually stand for identifier, is that what?

Cox: Yeah. Mm-hm.

Hsu: Okay.

Cox: Object identifier.

Hsu: Object identifier. Okay. When did your—when did the research group that you were in start to come into conflict with the larger Frame organization at ITT?

Cox: Oh, gosh. This is really more of Tom's history than my history.

Hsu: Okay.

Cox: But, Tom was deeply committed to this UNIX idea.

Hsu: Mm-hm.

Cox: And the rest of the Frame organization was deeply committed to IBM products.

Hsu: Okay.

Cox: And I think that—that split just eventually caused Tom to leave.

Hsu: Okay. <laughs> And how—so how closely did you work with Tom? Like were you—how closely did you collaborate?

Cox: Oh, very closely. But on different things. I was mainly focused on building that pre-compiler.

Hsu: Mm-hm.

Cox: He was mainly focused on getting UNIX adopted inside the organization. Very different.

Hsu: Oh, okay. What was his role in Objective-C? Like, did he have any design inputs, or did he help implement any of it?

Cox: Well, at that time he was very supportive. He—

Hsu: But it was mostly you were the one working on it?

Cox: Yeah. I was focused on the language and he was focused on providing an environment for the language.

Hsu: Mm-hm.

Cox: Remember, the pre-compiler was two jobs before Objective-C.

Hsu: Right. Okay.

Cox: So this was during the ITT period.

Hsu: Right.

Cox: And he—when he left he moved to Schlumberger-Doll Research Labs and I eventually followed him there. And then we both left to ~~form~~ [found PPI (Productivity Products International), which was eventually renamed Stepstone, and its first product,] Objective-C.

Hsu: Right. So how soon did you join him at Schlumberger-Doll after he left?

Cox: It wasn't immediately, but I would say months.

Hsu: Oh, okay. And it was for pretty much the same reason that, I mean once he was gone, you didn't see any reason to continue working at ITT, or you didn't see any support for UNIX within the organization?

Cox: Probably, but I honestly don't recall.

<Laughter>

Hsu: So you just wanted to just keep working with Tom?

Cox: Yeah, I think so. Schlumberger-Doll seemed like a nice place.

Hsu: Okay. <laughs>

Cox: I mean it was gold-plated. It was really nice.

Hsu: Wow.

Cox: <coughs>

Hsu: So, was there any particular reason why Tom joined that company?

Cox: Nothing comes to mind except they were local.

Hsu: Oh, okay, in Connecticut.

Cox: Yeah. And very well-respected. Their whole thing was applying artificial intelligence to oil field problems, oil field services. So everything was top-of-the-line AI kind of work. I think they had some interest in—in bringing in UNIX and, of course, Tom and I got all involved in that.

Hsu: Oh, okay.

Cox: Then Tom managed to eke out a Smalltalk license from Xerox, somehow.

<Laughter>

Cox: And he became Schlumberger's Smalltalk programmer.

Hsu: Okay. <laughs> And what—so what else did you work on there? Or were you continuing to work on the pre-compiler, or was that, or did you just stop doing that?

Cox: No. I didn't work on that there. They were working on—the Lisp group was very much involved in efforts to standardize an objective-oriented extension to Lisp.

Hsu: Mm-hm.

Cox: They were working with Danny Bobrow, and other folks [at Xerox PARC] to produce what eventually turned out to be CLOS.

Hsu: Okay.

Cox: Common Lisp, Lisp ~~Operating~~ [Object] System. What I was doing, I say—I would say supporting the Schlumberger's interest in UNIX.

Hsu: Mm-hm.

Cox: So there was—they had this very gold-plated, ginormous display console for displaying oil field data that was driven from VAX/VMS, as I recall.

Hsu: Mm-hm.

Cox: And we built this adapter. No, the big thing about that project was, what was it called back then? It was predecessor to Ethernet. One of these first generation Ethernet—

Hsu: Local area network?

Cox: Yeah, but very first generation.

Hsu: Token-ring? Okay.

Cox: It had a name, but I forgot that now. But anyway, we—proving that network as a way of connecting the display console with the computational part—

Hsu: Okay.

Cox: —projects of that nature that none of them really stand out in memory anymore.

Hsu: Right. So how did you and Tom decide to start your own company?

Cox: He got—I don't know how he made this contact, but he made a contact with someone in Philips in Europe that needed a worldwide review of their hardware engineering operation with an idea of productivity again, [applying object-oriented principles to their hardware simulation work.]

Hsu: Mm-hm.

Cox: And eventually, that contact culminated in a contract to do a worldwide review, so we had to ~~be away from~~ [leave] Schlumberger to do—to take on that ~~contract~~ [work], and—.

Hsu: Oh.

Cox: —so we both—

Hsu: So the—

Cox: —did that.

Hsu: Oh, so you guys became contractors, basically.

Cox: No, we started a contracting company.

Hsu: Oh, you—okay, you started a company, then you couldn't—okay, so the contact at Philips—so Philips wanted to con—wanted you to do this thing for them that as employees of Schlumberger you couldn't do.

Cox: Yes.

Hsu: And so you decided to just branch off and do your own thing—

Cox: Right.

Hsu: —partly because of these contracts.

Cox: Right.

Hsu: Okay. And so what was—so how—well okay, obviously, “productivity” is in the name of the company, Productivity Products International. <laughs> Seems pretty straightforward, <laughs> the name?

Cox: Oh, yeah.

Hsu: <laughs> So what was the vision? What was the business plan? What were you going to be selling?

Cox: Short answer is productivity. Now, to make that concrete, one concrete example of it is [a] more productive programming language.

Hsu: Mm-hm. So productivity—

Cox: The second—

Hsu: —in the sense of writing software.

Cox: Right. My answer to—for the second in that list was component software, software components.

Hsu: Okay.

Cox: So, the rhetoric that we came up with was, first, we'll produce [a language which serves as] the soldering iron—

Hsu: Mm-hm.

Cox: —and then we'll produce software ICs that can be joined together with that soldiering iron. Now that was the answer that we used back then. And then of course, that also we were also a consulting company.

Hsu: Okay.

Cox: So productivity consulting was what we actually did.

Hsu: Oh, okay. So trying to help other—other companies improve their software productivity?

Cox: Right. Or hardware. Philips was a hardware company.

Hsu: Oh, okay.

Cox: But their whole idea is they believed object-oriented approaches could help with hardware. So that's why they brought us in particular.

Hsu: And, oh that's interesting. So how did that apply?

Cox: I never fully understood [how that applied in practice but it was definitely their goal.]

Hsu: <laughs>

Cox: But they had a huge investment in software tools, which I did understand.

Hsu: Mm-hm.

Cox: And were written in general ~~hard~~ [problematic] languages, like Fortran, and I understood the problems with that.

Hsu: Mm-hm.

Cox: So it wasn't hard to make a claim that object-oriented approaches might help with that.

Hsu: Okay. I want to get more into the soldering iron metaphor. But before I do that, I think I read somewhere that Gerald Weinberg, the author of "The Psychology of Computer Programming", was doing some work for that Philips contract. How did that happen?

Cox: Was he involved in that? He was involved in several of our projects, but I don't recall him working on Philips.

Hsu: Oh, projects for ITT?

Cox: No. For other customers—

Hsu: Oh, after PPI?

Cox: —after we formed PPI. Yeah.

Hsu: Okay. Oh so—oh so, he was—he was sort of one of the people that you sometimes contracted with as part of [your contracting work]?

Cox: Right. It may have been Philips. I—it's just by that time, I—I was 100 percent heads-down working on the language side of the problem and not tracking the consulting side.

Hsu: Oh, okay.

Cox: So I—I might just not have—might not know.

Hsu: Okay. How did he end up being one of the people that you used as a contractor?

Cox: Tom had known him some—from previously. Then, of course, he—well, he was sort of the all-around advisor for the company.

Hsu: Okay.

Cox: I don't remember how—how that came about, but he was very helpful and very supportive.

Hsu: Mm-hm. Did his work or his ideas influence you or the company?

Cox: Well, certainly his ideas about effective consulting—

Hsu: Okay.

Cox: —were probably the most lasting. He also was a big help in helping me get my first book out.

Hsu: Ah, the *Object-Oriented Programming: An Evolutionary Approach* book?

Cox: Yeah.

Hsu: Yeah.

Cox: You know, negotiation kind of help.

Hsu: With publishers.

Cox: Help with publishers and so forth. He was a good friend. <laughs>

Hsu: Why did you seize on components as the solution to productivity issues? Software productivity?

Cox: I—because I can't think of anything more effective at achieving productivity than—than being able to lift it from here and put it down there—

Hsu: Mm-hm.

Cox: —a component, if you will. That much of it always made perfect sense to me.

Hsu: Mm-hm.

Cox: Now of course, in my opinion, industry has not followed that path—

Hsu: Mm-hm.

Cox: —with any—with any vigor. Well, I don't want to overstate that. There are—computer applications are components, even more so than objects. They're—and industry has followed that very aggressively, and now there's a lot more of interest in service-oriented architecture where components are—are very prominent. So I don't want to overstate it, but—- but in terms of the granularity of Objective-C objects—

Hsu: Right.

Cox: —industry is not following that.

Hsu: Right. Where did you first—where did you first discover the idea of component software?

Cox: Well, that was the ideal in selling Smalltalk.

Hsu: Oh, okay. So, but you saw encapsulation being, being components?

Cox: Right. This is [a] membrane around an object.

Hsu: Right. And where did the—where did that language, that specific term “component,” “software component” come from?

Cox: I may—it maybe coined or I may have heard, there was—there was at one time an academic interest in component C—, it had a three-letter acronym, C something. Component-based software probably, CBS. But I don't recall reading about it. It was just a term that captured the ideal that we saw.

Hsu: Okay. I want to get back to the soldering iron in the software IC metaphor. Where did that metaphor come from?

Cox: I coined it.

Hsu: <laughs>

Cox: It just captured what I liked about the Objective-C approach and what set it apart from the C++ approach.

Hsu: Which was, you've talked about C++ as more of a fabrication plant?

Cox: Right. It's very labor-intensive, highly efficient, very compressed development compared to the arm's-length approach that you have in Objective-C, the soldering iron approach.

Hsu: Right. But why use these sort of hardware metaphors? Like, why not use other metaphors? Why not use biological metaphors? Why was the—sort of this hardware metaphor so important?

Cox: I—it just seemed more approachable than biological. I mean one that would be more understandable to everybody.

Hsu: To computer people?

Cox: Well, everybody literally. But, yes, computer people.

Hsu: Okay.

Cox: I don't—I don't recall ever giving it much thought. It just seemed like the right <laughs> the right metaphor to use.

Hsu: I see. Okay. So then where did the name Objective-C originate? And when did you start using it?

Cox: It wasn't until PPI days. And then we needed a product name and that one popped out.

Hsu: Okay.

Cox: <laughs>

Hsu: So you—

Cox: It was coined like so many things.

Hsu: Okay. And so when you formed PPI you had already decided that the language was going to be one of your products?

Cox: Yeah. Yeah. [That was part of the plan. Consulting work to support building a products company.]

Hsu: Even when the contracting was going on, you were working on the language? Because that was a departure from—

Cox: Yeah. While we're—I remember while we were at Schlumberger, we were talking about what next after the kind of consulting contract, what products would we build.

Hsu: Mm-hm.

Cox: And ~~we~~ [Tom of course] knew about the pre-compiler that I worked on at ITT and I said, "Why not that? Why not turn that into a product"?

Hsu: Mm-hm.

Cox: So it must've been Schlumberger days.

Hsu: Okay.

<Laughter>

Hsu: And so what did that involve, turning the pre-compiler into a full-blown language—into a real language?

Cox: Multiple iterations. Remember, I had no background in language design, so I had to train myself on how to do that.

Hsu: Mm-hm.

Cox: And the first attempts weren't pretty.

<Laughter>

Cox: And so we just iterated the problem until we came out with a pretty good tool.

Hsu: I see. And in keeping with the soldering iron approach, it was basically just minimal, just the minimum amount necessary to create a Smalltalk-like environment on top of C?

Cox: The earlier versions were totally minimal. In fact, OOPC is the early version.

Hsu: Right.

Cox: That—big portions of that were originally a bunch of UNIX shell scripts.

Hsu: Oh, wow. <laughs>

Cox: And then, of course, as we went down that trail at PPI, I got spun up on tools like Lex and Yacc.

Hsu: Actual parser tools.

Cox: Right. And then got better and better. Eventually, when we got investment we brought in people that actually had a background in doing language design and they knew what they were doing.

Hsu: Mm-hm. Okay. I think Tom Love said that it was his idea to use square brackets in Objective-C, and for the language to have sort of this, the two levels. The object-oriented Smalltalk level and then the C procedural level. Is that correct?

Cox: I'm not attached to who claims that idea. [As I recall, I iterated over several possibilities and chose characters that didn’t trigger parsing conflicts in Lex/Yacc.]

Hsu: <laughs> Okay.

Cox: And I know most of the characters and I know how I chose specific characters for.

Hsu: Oh.

Cox: ~~And that was to try~~ [I just tried] characters like the square bracket and ~~see~~ [saw] if Yacc complained.

Hsu: Oh, okay. <laughs>

Cox: So—

Hsu: It wasn't because it was used for blocks in Smalltalk that you saw a similarity there?

Cox: I may have noticed a similarity, but literally the reason is, what can be parsed?

Hsu: Right. What didn't conflict with C.

Cox: Right.

Hsu: Right. So how much—what was Tom's contribution to ~~the~~ [Objective-C]?

Cox: Well, he was building the company and doing [most of] the consulting [business.]

Hsu: Okay.

Cox: —largely. I was doing the language.

Hsu: Okay.

Cox: Pretty cleanly spread around that.

Hsu: Right. Okay. And you were developing the language on a 512K Mac—Macintosh? <laughs>

Cox: <laughs> No.

Hsu: No. Okay. That's not correct.

Cox: I think that refers to Tom's first Macintosh.

Hsu: Oh, okay.

Cox: The—I did most of the development on [an early Unix] computer ~~called the~~ [built by] Fortune, [Inc.] It was [one of] the first desktop ~~LINUX~~, UNIX [machines.]

Hsu: Oh, desktop UNIX.

Cox: —that I know of, except for the earlier machine I referred to as the size of a coffee table. That was built by Onyx, an even earlier effort.

Hsu: Okay. So it was a workstation.

Cox: So the first was an Onyx, then a few years later came this Fortune desktop system. Little [five inch] floppy drives for disks is all it had [for storage.] Pretty primitive, but that's what I used [until we moved to Sun workstations much later.]

Hsu: Okay. So was Objective-C at this point, I guess you mentioned there were two stages. At what point was it a full compiler, or was it like a macro preprocessor at some point? Where they just compiled into C and then compile the C back into whatever native—is on the machine?

Cox: I would characterize it as a process of steady improvement.

Hsu: Okay.

Cox: But the time it was—it became clearly a compiler with no apologies was when we got investment and brought in a programming staff.

Hsu: Oh.

Cox: And they did a careful design of the thing. [There were] probably a half a dozen versions getting to that point.

Hsu: Okay.

Cox: But, you know, just getting better, but not perfect.

Hsu: Right. Okay. So it could've been in that intermediate stage it could've been a macro preprocessor.

Cox: No, not a macro preprocessor. Parsing those message expressions, it takes more than a macro can do.

Hsu: Oh, okay. Okay.

Cox: But simpler, a simple approach—

Hsu: Right.

Cox: —was where we started and got better over time.

Hsu: Right. Okay. So the—so you mentioned—let's talk about when you started to get venture funding. What was the trigger for that?

Cox: The trigger?

Hsu: Or at what point did you and Tom realize that you needed to get external funding?

Cox: Well, we, I think from day one we knew that was part of the plan.

Hsu: Oh, okay.

Cox: And so in parallel with that consulting contract and development of the compiler, we contacted venture capital firms and it eventually culminated in investment.

Hsu: Okay.

Cox: I think what they saw in the company was not—I think what attracted them was this idea of componentry, not the specific tool.

Hsu: Mm-hm.

Cox: They were pretty clearly not interested in supporting pure language development. They had some history with doing language funding. But they liked the idea of components.

Hsu: Okay. And so that was the reason. So it was at that point that you started to hire people with experience designing programming languages.

Cox: Mm-hm.

Hsu: Okay. Was that around the time that you hired Steve Naroff?

Cox: Yeah, he was one of those.

Hsu: Okay. What was it about Steve Naroff in particular that attract—that you thought he was a good person for the job?

Cox: I don't know, I just liked the guy.

<Laughter>

Cox: I don't recall anything in particular.

Hsu: Mm-hm. Was it his previous experience?

Cox: Well, he had some experience with SAIL that seemed—that seemed relevant.

Hsu: Okay. Okay. Let's see. So the company was in Sandy Hook?

Cox: Yeah.

Hsu: Or was it originally in Sandy Hook or did you move to Sandy Hook?

Cox: We moved. Before that it was nearby, but in an old dentist office.

Hsu: Oh.

Cox: You know, startup quarters. I think it was in the same town, but I honestly don't remember for sure.

Hsu: Okay. And then the new location after you expanded was in an old waterfall-driven factory?

Cox: Mm-hm.

Hsu: <laughs>

Cox: It's a beautiful place.

Hsu: Yeah. Did that have anything to do with the colonial era metaphors that you were using?

Cox: Not really. But I had taken to, I don't—I was very taken by the idea of interchangeable parts in musket manufacturing.

Hsu: Mm-hm.

Cox: Because that seemed to express this componentry, component idea very well. But I don't think that connected with the Sandy Hook factory.

Hsu: Okay. What was it about colonial era, gun manufacture in particular? Was it—I mean that's a very interesting histor—particular historical moment. What—why did that speak to you so much?

Cox: The idea of a gun part, like a trigger that can be taken out of this gun and put in 100 others and work just as well, interchangeable parts, seemed to be the birth of the componentry idea and something that could be applied to software pretty straightforwardly.

Hsu: Mm-hm.

Cox: So, you know, you know, it's hard to reconstruct why you're drawn to particular analogies in retrospect, they just click on you and you build on them.

Hsu: Mm-hm.

Cox: But I was trying to get across this idea of prod—ways software development could be more productive.

Hsu: Mm-hm.

Cox: Because it never seemed to me that language alone would make much difference. Switching from C to Ada would be nothing.

Hsu: Mm-hm.

Cox: From C to C++, it seemed to me to only have marginal changes, marginal improvements. So what could bring major improvements? Components.

Hsu: Right. Meaning revolutionary improvement.

Cox: Yeah.

[Break]

Hsu: Right? All right, we're back. That was actually very interesting, the story that your wife told.

Cox: <laughs>

Hsu: Why don't we talk about how you met your wife and, yeah.

Cox: Oh sure. Well, this goes even further back to University of Chicago days. At the time I was very much into bluegrass playing—

Hsu: <laughs>

Cox: —guitar playing. And Bill Monroe was the father of bluegrass, had a farm in [Beanblossom] Illinois, I think it was. And he held a convention every year, bluegrass convention. So I went down there [mainly] to pick guitar around the campfire with the people who paid to be there.

Hsu: Mm-hm.

Cox: But and also to watch the performances. And she was there, oh gosh; she was there with, alone or with someone. I don't—that part I don't remember. But anyway, she had this little Yorkshire terrier that attracted my attention and one thing led to another from there.

<laughter>

Hsu: And when were you married?

Cox: Just about the time I left [Chicago.]

Hsu: Nineteen—

Cox: When I left Chicago.

Hsu: Oh, oh, Chicago.

Cox: Right.

Hsu: So that was, so what year was that?

Cox: Oh, you'd have to drive—it would've been in seventies sometime.

Hsu: Seventies? <laughs>

Cox: Mm-hm.

Hsu: And, I mean she just mentioned that she decided to get a degree in was it Management Information Science, MIS?

Cox: Well, she has several degrees, Master degrees. Etta, what—what was—were you going for a degree with that Cobol course or?

Glenn: Yeah. I remember <inaudible> get another Masters in Computer Information Systems? It's not like my whole life just collecting <inaudible>. <laughs>

Hsu: <laughs>

Cox: Yeah.

Glenn: One.

Hsu: So—

Cox: So this was much later, of course.

Hsu: Okay. Right. Right. So at the time that—so this was when you had already formed PPI.

Cox: Mm-hm. Right.

Hsu: Yeah. Did she ever help—did she ever help with the company at any point?

Cox: Yeah.

Glenn: Yeah. <inaudible>. As a matter of fact, during that time. Should I talk?

Hsu: Sure. If you want, you can come over.

Glenn: Oh, no, that's fine.

Hsu: <laughs>

Glenn: During that time we, you know, as I mentioned earlier, we were like little kids and were so excited because we were starting a new company, and so the wives, they helped wherever they could.

Hsu: <laughs>

Glenn: It was a fun time. And to have your, you know, telephone ring at home. "Hello"? "Steve Jobs here. Need to talk to Brad".

<Laughter>

Glenn: "Okay".

<Laughter>

Glenn: My name is Etta Glenn. I'm Brad Cox's wife. And we were talking earlier about how—how it was at the very beginning when we started PPI and our International—Productivity Products International. That was the first name for the company. But we were like little kids. You know, we were—wherever you could help, you know, we pitched in. And it was a wonderful time. I can remember when we moved into this dentist office that Brad mentioned earlier, we had a rather substantial contract with Philips. Do you recall that?

Cox: Who?

Glenn: Philips. I thought it was Emerson—we had. Okay. <laughs> And we were—we were in a dentist's office. And each office had a sink, you know, typical dentist office. And so the big honcho flew over from Europe that was with Philips, and we didn't know what to do with him. So what happened is someone met him at the airport and we made arrangements for a very nice hotel in New York City, where he stayed. And to keep him from coming to the company and to see how <laughs> how it was, we kept him in New York watching plays and going out for dinner. But it all worked out. I can't help but imagine he thought this was really a very strange thing. But it was fun.

Hsu: Mm-hm.

Glenn: It really was.

Hsu: <laughs>

Glenn: And then, like I said, to have Steve Jobs call and say, "Hello, got to talk to Brad now", you know? <laughs>

Hsu: <laughs>

Glenn: So.

Hsu: So your wife mentioned that Apple gave you some computers?

Cox: I may seem really out of touch here, but that I don't recall.

Hsu: Uh-huh.

Cox: I don't recall actually having Apple or NeXT computers on hand, except Tom's Macintosh that he was doing Smalltalk on. That must've been come after the sale of Objective-C rights to Apple.

Hsu: Oh, okay.

Cox: That's—that's the only way that makes sense technically is we transferred the rights to Apple when they finished the development.

Hsu: So post-1997.

Cox: Right.

Hsu: I see. Okay. Let's—well, let's back up and talk about how did your company get involved with NeXT and Steve Jobs.

Cox: Okay. Remember, I was head down—heads down working on component development at the time and only indirectly involved in the actual negotiations. But, as my memory—from memory there were two main candidates for an objective-oriented language for NeXT, Object Pascal and Objective-C.

Hsu: Mm-hm.

Cox: And they both had big supportive factions, but—

Hsu: Within NeXT?

Cox: Yeah. But Objective-C turned out to have the advantage in that its base was available on everywhere. It was more ubiquitous. Pascal was harder to find. And so kind of back and forth over an extended period of time, but eventually they settled on Objective-C.

Hsu: I see.

Cox: I think Steve [Naroff] talked about some of the reasons in that paper we were looking at earlier.

Hsu: Steve Naroff you mean?

Cox: Yeah.

Hsu: Okay.

Cox: And, of course, much of the details there were invisible to us. We were, you know, <inaudible> on the way.

Hsu: Mm-hm.

Cox: But that's what I picked up over time.

Hsu: I see. But they could've chosen C++. If they wanted a C base, why did they go with Objective-C over C++?

Cox: It's just totally un—what they wanted.

Hsu: Uh-huh.

Cox: Remember that soldering iron analogy?

Hsu: Right.

Cox: That ideal is actually how Apple's—

Hsu: Or NeXT.

Cox: —business model has developed. They basically take—write software components to do things like scroll bars, for example, [and everyone else uses the components they provide.]

Hsu: Mm-hm.

Cox: Provide it with a—[Using Objective-C as the] soldering iron.

Hsu: Right.

Cox: —to their end users and their end users glue these components together to build iPhone applications.

Hsu: Right.

Cox: So it's—it's directly software-IC based.

Hsu: Mm-hm.

Cox: The only difficult—The dichotomy there is that we original—PPI originally saw ourselves as components builders and Apple saw themselves as in that role [also].

Hsu: Okay.

Cox: So that's where—

Hsu: There was a conflict of interest.

Cox: Right.

Hsu: Right. Okay. So you didn't have any direct—how much contact did you have with Steve Jobs?

Cox: Very little.

Hsu: Very little.

Cox: Met him at a conference once.

Hsu: Okay.

Cox: But that [was] the extent of my involvement. I was pretty heads down on components—

Hsu: On components.

Cox: —that whole period.

Hsu: Right. Well, let's go back and talk about that. So, you know, so at this point in time—well, so actually, let's go back even further. Why did the company's name change to Stepstone after you got venture capital funding?

Cox: The venture capitalists wanted [a new name.]

Hsu: <laughs> Okay.

Cox: I don't—I never understood why they didn't like PPI, but they—they wanted a name change.

Hsu: <laughs> Okay. And so where did the name Stepstone come from?

Cox: Oh, like everything else, it was coined, the idea of stepping stones.

Hsu: By yourself or by Tom, or by one of the VCs?

Cox: Oh, we probably had the typical naming contest internally.

Hsu: <laughs>

Cox: You know, nothing very entertaining to talk about, just a name that seemed to fit.

Hsu: I see. So the idea of stepping?

Cox: Productivity again.

Hsu: Right.

Cox: Step stone to productivity.

Hsu: Stepping stone to productivity.

Cox: Yeah.

Hsu: Okay. And so you mentioned that your focus shifted from the language to the libraries. But it seemed like at this point you sort of gave Steve Naroff full run of the language.

Cox: Well, while he was at Stepstone he was part of the language development team.

Hsu: Mm-hm.

Cox: And they're responsible for the language from that point on. The library initial development, that was what I was doing.

Hsu: Okay. We were talking about the language work, the interpreter, the garbage collector. So you mentioned that that was going on in parallel with the libraries?

Cox: Right. So it was a continual pull from all kind of directions, from sales, from customers, from marketing—

Hsu: Mm-hm.

Cox: —to add this feature or that from—from all directions. Garbage collection was one of the big ones. Smalltalk blocks was another—was certainly something I was very interested in and worked on very hard.

Hsu: Right.

Cox: And the interpreter was an even more complicated issue. Because compared to C an interpreter is huge. It's a huge development there. One of our customers built one—built an interpreter, very poorly structured in my opinion, un-maintainable thing. So working on the interpreter was basically turning that into something we could call a product, which is a huge project because of its size.

Hsu: Mm-hm.

Cox: And I was never very happy with the result. It could do little things okay, but anything more than a couple of lines, you know, wheels start falling off pretty quick.

Hsu: <laughs>

Cox: <laughs>

Hsu: So none, so those three, blocks, the interpreter, the garbage collector, how far did you—did the company get in pursuing those features? Were they ever shipped?

Cox: Yeah. I—the blocks was the one that was most feasible. The other two were frankly, unfeasible in my opinion, but there was a huge demand for them.

Hsu: Mm-hm.

Cox: But blocks was eventually turned into something called ICPak 201, which was—

Hsu: So it was in a library.

Cox: Yeah. Which was the foundation for things like, not just blocks, but asynchronous tools—

Hsu: Oh.

Cox: —queues and—

Hsu: Concurrency.

Cox: —Concurrency and a bunch of user interface stuff was in there.

<crew talk>

Hsu: So you mentioned that the company's strategy at this point in time was on the software ICPaks, the libraries, and that caused a conflict of interest with one of your customers, NeXT, which was also building their own libraries. Was that business decision coming from the venture capitalists that had invested in the company or?

Cox: Well, I wouldn't—I think calling it a conflict of interest is overstating it. It's just a difference in point of view.

Hsu: Mm-hm.

Cox: The investors liked the components approach and supported it. I don't know that ~~Apple~~ [NeXT] thought of themselves as being in the components business. They didn't articulate things that way, but that maybe my view of what business they were actually—I've always thought of Apple as being the—Brad's way of thinking about Apple was as a software components company.

Hsu: Okay. Meaning NeXT.

Cox: I'm not sure if they would agree with that or not, but.

Hsu: Right. Meaning NeXT was a software components company.

Cox: NeXT and then Apple.

Hsu: Yeah. And then Apple. Right.

Cox: Yeah.

Hsu: I think to some extent that's accurate. I did an interview with Steve Naroff and he said that Steve Jobs had—there was a meeting. There was a meeting with—between Steve Naroff, Tom Love on the Stepstone side, and Steve Jobs and Bud Tribble on the NeXT side where Jobs told Tom to stop focusing on the ICPaks and make Objective-C, the language, great because he didn't think that—he thought that NeXT's own libraries were better than your libraries. What did you think? —Were you aware of this?

Cox: Not at the time. I mean I—only when Steve [Naroff] and I worked on that paper is the first time I heard of that.

Hsu: Mm-hm.

Cox: But again, I was not in that meeting.

Hsu: Right. Would you disagree with that statement?

Cox: No. Not at all. Because the thing I've never managed to successfully do was to find a business model for these components. I mean they're made of bits, see? They can copied in nanoseconds.

Hsu: Right.

Cox: So you can't buy and sell them like a can of sardines.

Hsu: Mm-hm.

Cox: But Apple could because they could nail all these software things down to a tangible piece of hardware and build a business around that.

Hsu: Right.

Cox: So, you know, and time proved them right.

Hsu: Right. Right. I'll get to that later. But I wanted to go back and ask, so you mentioned that Objective-C's object-oriented layer was dynamically typed and only had static typing at the procedural C layer. And Naroff started to add some static typing in the higher layers.

Cox: Mm-hm.

Hsu: Was that a good thing in your mind, or did that take away from what your initial vision was for the language?

Cox: I always thought of it as just improvements—

Hsu: Mm-hm.

Cox: —you know, I was glad to have. But anything that improved on what C provided was an improvement.

Hsu: Mm-hm.

Cox: It's not where I spent my own time, but Steve had his own ideas about where to make improvements and I didn't really disagree.

Hsu: What do you think of sort of these constant debates between static and dynamic typing and binding?

Cox: I've—since those times I've largely almost—I've entirely switched over to Java because of—and I've grown to really appreciate the kind of static typing that it provides.

Hsu: Mm-hm.

Cox: The C++ approach always struck me as toxic for sharing all these header files and oh, what's that term for it? Just seemed like not a move toward more productive language in most respects, but Java did. It had garbage collection, it had very supportive IDEs. It was, you know, a breath of fresh air. So I've gone—I've gone to Java and never looked back.

Hsu: Okay.

Cox: I forgot the question. <laughs>

Hsu: Right. So it's not, I mean you're sort of agnostic in between static and—static and dynamic typing or static and dynamic binding. I mean Java has static typing but dynamic binding.

Cox: Well, when static does the job it's the tool to use, but it doesn't do all jobs. So if you can check errors early at compile time, why not?

Hsu: Mm-hm. Right. You mentioned that your interview for the book *Masters of Programming*, that if you had to start over again, you would remove inheritance from Objective-C because it's not that important. Why?

Cox: Well, that's an overstatement, but not far from the truth.

Hsu: <laughs>

Cox: At one time I thought it was important, mainly because Smalltalk had it and I was—my goal at that time was to simply duplicate as much of Smalltalk as I could. But since then I almost never use it anymore.

Hsu: Mm-hm. What's the problem with inheritance?

Cox: It's—it's rarely <coughs> I did all kind of experiments on trying to find the proper use, the right way to use inheritance and none of them were really very satisfactory. Because anything you put in the super class everything inherits it, and there's no way to control it. Plus, encapsulation can be used, I forget the name of the guy who wrote a whole paper about this, but basically showed how everything you could do with inheritance you could do with encapsulation. And then those Smalltalk derivative, I think called, what they call it, Squeak.

Hsu: Squeak or Self or Squeak? Yeah.

Cox: Self was—

Hsu: Self.

Cox: Yes.

Hsu: Prototype based language?

Cox: Yeah. That basically did without inheritance altogether and it—they eventually won me over. I tried really hard to find a way to use inheritance correctly, but I eventually, I just ~~quit~~ [gave up trying.]

Hsu: Okay.

<Laughter>

Hsu: Let's go back to talking about the NeXT contract. What did you think of the things that NeXT wanted to change in the Objective-C language?

Cox: At that time my focus was totally on components—

Hsu: Right.

Cox: —not on language. I don't recall having a firm opinion, okay?

Hsu: Oh, okay. <laughs> Was there—were you upset when Naroff decided to join NeXT and leave Stepstone?

Cox: Probably. Probably. I don't remember any great angst.

Hsu: Okay. <laughs> Did the NeXT version of Objective-C basically fork from the Stepstone version? Were there essentially two different versions of the language?

Cox: Well, there was a period of transition, but ultimately, the outcome of the transition was that Apple fully acquired rights—

Hsu: Right.

Cox: —to Objective-C. And so there was no Stepstone Objective-C work after that.

Hsu: Okay. That happened in what, '95? 1995 or so?

Cox: Plausibly.

Hsu: Okay. Well, how did that sale occur? I mean was it—I mean how—was it because of Stepstone's other financial difficulties or why did Stepstone decide to sell the rights of the language to NeXT?

Cox: I've never heard anyone answer that question directly. So probably it was just no other option by that time. Sorry, I can't be more specific.

Hsu: Okay. Was Stepstone ever involved in getting Objective-C into the GCC Compiler?

Cox: No.

Hsu: No. Okay. Okay. Naroff also mentioned that Hewlett-Packard also was a client that wanted to use Objective-C.

Cox: Mm-hm.

Hsu: What did they use it for?

Cox: Research. As a research language, they [were] big supporters of it.

Hsu: Mm-hm. So it was just inside the HP labs.

Cox: Yeah.

Hsu: So it never made it to any products?

Cox: Not to my knowledge.

Hsu: Okay.

Cox: It's not to say I wouldn't have knowledge of that, it's just I—I don't know.

Hsu: Right. Okay. Let's go back to the—your work on the ICPaks. So I think you've written that your focus on the ICPaks was using it for GUI, graphical user interface libraries.

Cox: That's one of two products.

Hsu: Oh, okay.

Cox: The first product was the foundational classes. The kind of thing that you immediately associate with Java, you know, sets and strings and, you know, the low level libraries.

Hsu: Right.

Cox: The other ICPak was the kind of things you get from Java with Swing.

Hsu: Yeah. The—

Cox: The GUI, the components.

Hsu: —classes.

Cox: So those were the two ICPak products at the time. Seem like there were more, but maybe I'm confusing the ICPak work with the interpreter and—

Hsu: Oh.

Cox: —there were a lot of projects going on in all three spaces, so it's—the boundaries between them are often a little sloppy.

Hsu: Right. I mean you mentioned that the blocks stuff ended up going into an ICPak that was later on use for threading and concurrency type stuff.

Cox: Right. I think that was—that was probably the next release—

Hsu: Oh, okay.

Cox: —that never—

Hsu: Never shipped?

Cox: —never shipped.

Hsu: But you did ship the GUI, the interface?

Cox: The first two libraries did ship—

Hsu: The first two libraries.

Cox: —and this was—this was the futuristic thing that never made it.

Hsu: Right. And why focus on the GUI part of it?

Cox: Because without that, that would seem to be the big answer to productivity, programmer productivity, was something like Smalltalk user interface.

Hsu: Mm-hm.

Cox: Just made total sense at the time. I'm not sure I believe it now, but at the time it seemed very plausible.

Hsu: Right. I think you wrote in that paper actually, that the focus on the graphics libraries ended up bogging the company down because you have to try to make it work on every individual platform?

Cox: Well, it was a heavy weight to haul around. The language itself could be ported in hours but this graphic library was very heavyweight to port. Now whether that drags the company down, I'm not sure I'd go that far because [a portable GUI solution] was also very attractive in those days.

Hsu: Right.

Cox: So it was a big selling point, it's just expensive as all daylight to do [well when the underlying framework to build on is proprietary to every platform.]

Hsu: So it was a big expense for the company, but it was also generating a lot of revenue for the company.

Cox: Mm-hm. Right.

Hsu: How much of the—so was most of the company's revenue derived from selling the ICPaks? Or <inaudible>?

Cox: I don't remember the split between language sales and the two library sales.

Hsu: Oh, okay. But it was both. It was either one or the other.

Cox: Right. As I recall the ICPak, one of them was bundled with the compiler.

Hsu: Oh, okay.

Cox: As I recall the IC—the user interface library was separate.

Hsu: Right.

Cox: And I don't remember the split between them.

Hsu: Oh, okay. But if you added them together they would be 100 percent, or did you still have contracting revenues?

Cox: There was occasional contracting revenue. Nothing as big as the Philips contract.

Hsu: Okay. How did—talk about the sort of—what do you think [are] the reasons why Stepstone didn't succeed in the end?

Cox: Well, remember this was so early that there were no answers to how—nobody knew how the world was going to unfold back then.

Hsu: Mm-hm.

Cox: So you're making your best guesses and you're placing your bets. At the time, and throughout the history, our focus was on programmer productivity.

Hsu: Right.

Cox: I never imagined that this open-source movement would come along—

Hsu: Right.

Cox: —where programmers are essentially [viewed as] free.

Hsu: Right.

Cox: I couldn't have predicted the world would unfold in that way—

Hsu: Right.

Cox: —and still don't fully understand how it, you know, what supports the skyhook.

<Laughter>

Cox: But, so be it. That's how things transpired and—

Hsu: Right. So essentially, the whole business model was obsoleted by open source. Because some people could create libraries for free and your business model was based on selling one.

Cox: That's the claim anyway, that they can create libraries for free. I don't buy the claim because somebody, I mean they have to support themselves somehow.

Hsu: Right.

Cox: It's just I don't fully understand how they do.

<Laughter>

Cox: It seems like black magic.

Hsu: Yeah. Well, let's go back to the publication of, one, your book, but also the two articles that you're famous for. One was the IEEE Software Engineering article—"Software Revolution" article, and then the other one was the 1991 *Byte Magazine* article, "There Is A Silver Bullet", which—

Cox: Oh, yeah.

Hsu: —can you talk about how you came to publish those?

Cox: We're trying to articulate what we were just talking about.

Hsu: Right.

Cox: This view that—that there could be a component market.

Hsu: Mm-hm.

Cox: You know, I repeat again, so little was known back then about how the future was going to unfold. Like evil people hadn't been invented yet.

Hsu: <laughs>

Cox: I mean we were building email systems that—where you could forge the sender address.

Hsu: Mm-hm.

Cox: I mean that—that's what email is to this day. So much hadn't been invented yet. And I was very eager to define—to help push for a future in which people could earn a living building components.

Hsu: Mm-hm.

Cox: And so both of those articles were trying to articulate that hey, guys, we could go this way. Like the future could look like this.

Hsu: Yeah.

Cox: And not a future that was supported by advertising and malware.

Hsu: <laughs> Right.

Cox: Which is what the industry effectively chose.

Hsu: Right.

Cox: So I—I gave it my best shot, but I don't feel like it was anywhere near successful.

Hsu: But it was, I mean pretty provocative that, I mean the title of your *Byte Magazine* article is an explicit rebuttal of one of Fred Brook's well-known articles, like "[No] Silver Bullet".

Cox: Mm-hm. It could—I think yeah, I stand by that.

Hsu: It was—it was very deliberate.

Cox: Yeah.

Hsu: Yeah.

Cox: And I always—I never viewed it as a challenge to Fred Brooks. I viewed it as standing on his shoulders. He articulated that—that thing but hey, we can polish these edges and get, you know, closer to where he's trying to get to.

Hsu: Right.

Cox: Because that idea of off-the-shelf components could be the silver bullet for software productivity, if only we could find a business model for selling them. And so I then focused on the “if only” part of that, that clause.

Hsu: Right. Right. You explicitly talk about Thomas Kuhn and his book, *The Structure of Scientific Revolutions*, and the notion of paradigm shift. Where did you come across that and why did you, why was it useful to you to cite?

Cox: Oh, gosh. I don’t remember the context now. Do you recall?

Hsu: You use it to say just like, similarly, when the Copernican Revolution and the Industrial Revolution—components, software components, are sort of the Software Industrial Revolution. And this is like a Kuhnian paradigm shift. And you likened it to that.

Cox: I’m sorry. I’m not remembering immediately.

Hsu: It struck me because most computer science type articles don’t draw on history of science.

Cox: Exactly.

Hsu: Or history in particular. I mean, the colonial…

Cox: Well, that's the thing that sent me down this components trail so strongly, because how do you have a science when every component you meet is the first time you encounter it? There's no reusability, like steel. You saw steel last week, so you know what steel is this week. They're no bricks. There's no—there's not… So how do you have a science where everything is new every time you see it? Answer is you don't. I mean, so I say there's no such thing as computer science or software engineering.

Hsu: Right.

Cox: I've taken to using the analogy of mud masonry.

Hsu: Right.

Cox: So I don't know if Kuhn is useful in illustrating that point or not, but…

Hsu: <laughs>

Cox: I must've used him for some reason.

Hsu: At the time it was, you seem to have liked, using it. You thought it was a good idea? <laughs> Okay. So then, I mean, it struck me that, you know, you mentioned the business model, right? The business model. So that your idea for the market for components requires, well, a market. How important is it for the market to be part of the solution? Because it's not a traditional computer science thing to think of—that's usually [thought] of a lot as economics, right? Why is the market so important?

Cox: Because where else are you going to get components?

Hsu: Well, open source.

Cox: Well…

Hsu: <laughs>

Cox: Look at the components you get from open source. And that doesn't speak—the components I know about from open source, don't reek of quality. And don't reek of reuse. That epitomizes exactly what I was trying to avoid. Because to me a component is something that people obsess about for generations. You know, pour all of their energy into that particular component. It's their—what they hope to sell, in the market.

Hsu: Very high quality. And open source doesn't create the incentives to do that?

Cox: Well, I'm generally disappointed with what I get and I spend my, all of my time, doing, using, Spark for big data calculations, right? Spark is… It's the best idea going, but boy, it's ~~messy—~~[really a big mess of fragile code.]

Hsu: Right.

Cox: And it's hard to use that as an example of something to aspire to.

Hsu: Right. But isn't the idea of open source that if you don't like it you can go and fix it yourself rather than…

Cox: Sure. [If the roof collapses on your mud hut,] you're free to fix it yourself. [But wouldn't you have been better off to build it from commercial components like bricks?]

Hsu: <laughs>

Cox: But it's like if you're building a house, you're free to dig mud in your backyard and build a house [for free. Or do it with not-free components like bricks. Both are viable choices, but advanced societies generally have opted for bricks.] I mean, so…

Hsu: But it's not good for mass-producing houses.

Cox: Right. Or I would say even for houses in general. You generally get better quality if you take this other approach where you buy components from people that make their career around providing components.

Hsu: Right, right. There were actually a few companies in the NeXTSTEP space, on the NeXT platform, that did actually try to sell component libraries. But a lot of them didn't take off because the objects were either too specialized or they weren't well tested enough or they weren't well documented enough. Were you aware of this?

Cox: No. No, I wasn't.

Hsu: Yeah. They were called ObjectWare. There were a couple companies that were trying to sell them. Don't think they ever succeeded that well.

Cox: Yeah. I'm not surprised, but…

<laughter>

Cox: It turned out to be a very hard thing to do, because, again, how do you—what replaces scarcity [for] component[s] that can be copied in nanoseconds. What do you do to charge for [them? It's a very deep problem indeed.]

Hsu: Right. Well, that's a good segue to the book, your later book that you wrote, *Superdistribution*. So that, that book describes sort of this idea where you're moving away from selling the components to metering their usage. Did you get the idea from Japan? Where did that idea come from?

Cox: Well, I got the name from Japan.

Hsu: Okay.

Cox: There was a paper by that name, *Superdistribution*. The idea seems to be original. I remember a conversation with—oh, damn. Don't do that, brain. Ken. Ken Hamer-Hodges, in my office. Where I was trying to articulate what makes software ICs so different from hardware ICs. And that seemed to be the start of the idea of—they are fundamentally different things in opposite universes, you know. One abides by conservation of mass and the other doesn't. So they're more different than similar. And so how do you charge for something so different?

Hsu: Right.

Cox: So [selling them] by the use [instead of by the copy] came out of that conversation. Only problem was, [Objective-C objects] was the wrong level of granularity to attempt it. Apple eventually found an answer to that question of how to charge for it. You nail it to hardware and sell the hardware. And then in

[the] years since, SOA came along [with the subscription model, which made the idea quite viable. Pay by the month basically.]

Hsu: SOA?

Cox: Yeah.

Hsu: Service-oriented?

Cox: Service-oriented architecture.

Hsu: Architecture.

Cox: Now [since evolved into the] cloud. Where the answer is, build your components over yonder on that server and charge for the use. Don't ship copies. Just run it only there [and charge for how much people use.] So that's two workable answers to the question I posed of how to charge for the stuff [made of bits].

Hsu: Right.

Cox: "Superdistribution" was a third attempt to answer it, but the granularity was all wrong. [The basic problem is that objects of this granularity run on the computer that the end-user controls, so there's no real way to prevent them from interfering with your metering.]

Hsu: Right. Because you were still trying to do it at the granularity level of the object components, the same level that the software ICs were getting at, which was sort of the, I mean, you called it, what, the chip, the gate level?

Cox: Right.

Hsu: That was the metaphor that you used, the chip and gates.

Cox: Yeah.

Hsu: Whereas the ideal granularity is at the, what, the rack or the card level or…

Cox: Oh, I'd never say that there is a—I would never claim an ideal level. It's just those are three, those are, what was it, five levels that the hardware engineering community finds essential [and that I could correlate to similar levels in software.]

Hsu: Right.

Cox: Different tools for different jobs.

Hsu: Right. And so articulating components, software components, using the metaphor of hardware at the time made a lot of sense to outline these sort of granularities, but where the metaphor breaks down is that software can be copied infinitely and has no material scarcity.

Cox: Right.

Hsu: And so you have to find some other mechanism to—

Cox: To charge for it.

Hsu: —charge for it.

Cox: Yeah.

Hsu: Okay. So you left Stepstone in—is it 1990, after Tom Love?

Cox: Oh, yeah. After Tom. But don't pin me down on dates.

Hsu: Okay. <laughs>

Cox: I just, I don't retain them.

Hsu: Okay. <laughs> So Tom left because he had a disagreement with the board and the new CEO he appointed?

Cox: Right. And I don't remember exactly what about.

Hsu: Oh. <laughs> Okay.

Cox: I wasn't in on those conversations, so I can't shed much light. You'll have to ask him.

Hsu: Okay. <laughs> But once again, after he left, you didn't see much reason for you to stay or…

Cox: Well, it seemed like I was, I did stay, for an extended period, but I—well, I stayed until the company closed, basically.

Hsu: Okay. Right.

Cox: Which must've been, like, maybe as long as a year.

Hsu: Oh, okay. So how tough was that period for you?

Cox: Well, sales-wise it was terrible.

Hsu: <laughs>

Cox: But again, my job in those, at that time, was the research arm of the company, building future products. And I kept trudging away on that goal.

Hsu: Mm-hm. Was it because the market for the components suddenly evaporated? Because you were making decent revenue before.

Cox: I don't know. I wouldn't call it suddenly evaporated. It's just, it had been a hard haul all along.

Hsu: Okay.

Cox: And one I wasn't directly involved in, so I don't have much to add to it.

Hsu: Okay. So what did you do after you left Stepstone?

Cox: I worked on the second book for—

Hsu: Okay. The *Superdistribution* book.

Cox: Yeah. For quite a while. And then I joined George Mason.

Hsu: University. Okay. Is that when you moved from Connecticut to the DC area?

Cox: Yeah.

Hsu: Why did you decide to go back to academia?

Cox: I don't know. They called me. This was a spinoff from the Economics Department there. Was called Program on Social and Organizational Learning, PSOL, P-S-O-L.

Hsu: Right.

Cox: And this was basically the group that was, economics group, that was trying to articulate why capitalism is better than communism. I'm oversimplifying grossly here, but…

Hsu: <laughs>

Cox: And they saw my efforts in bringing software into the capitalism approach as being what they were trying to achieve more broadly. So that's why they brought me in. They saw some synergy there.

Hsu: That makes a lot of sense.

Cox: And they also liked the idea that I was a software developer. And George Mason was interested in building a way to make revenue out of software development.

Hsu: Oh, yeah. That's true. Because I know that they have a group there that makes software for the digital humanities and for history in particular. Were you involved in that at all or…

Cox: I know the guy who was building CDs for history. The revenue angle, I hadn't turned up before.

Hsu: Oh, like CD-ROMs?

Cox: Well, yeah. The idea that you could make money doing that hadn't been…

Hsu: Oh, okay.

Cox: Hadn't come up in my experience. In what context did you hear about it? In a money context or was it…

Hsu: No, no, no. From an academic context, actually. They have a program where they have graduate students doing research, developing software for humanities professors and history professors to do their own work. So one of the, you know, products that come out of it is a program called Zotero, which is like a digital reference, a digital library. You can use it to manage your citations, a citation manager. And can use it to automatically reformat all of your citations using a different, you know, from Chicago style to MLA or whatever. Just at a, you know, with a press of a button. But that, that product is also open source, so it's not a product for sale, but yeah. But they make these tools, and yeah. So I became familiar with it for that reason. So wait. You said that you view computer science as more of a social science than a physical science. Is that… That correct?

Cox: I think it'd be more accurate to say [it's my view that] it's not a science [at all.]

Hsu: Okay.

<laughter>

Hsu: Well, then what is it? <laughs> If it's not a science. Is it an art? Is it engineering? Is it mathematics?

Cox: Certainly not engineering.

Hsu: Ha.

Cox: Certainly not mathematics. Maybe an art would be, a craft, would be—

Hsu: A craft.

Cox: —a better term for it. But even a craft—There's reusability of know-how across materials. You know, leather today is leather yesterday. We don't have that. [Without that, how can it be either science or engineering?]

Hsu: Mm-hm. When you were at George Mason, did you read a lot of political economy? Adam Smith, Karl Marx, Austrian economists?

Cox: Well, all of that was very much in the air. My focus at George Mason very quickly turned to the internet.

Hsu: Oh.

Cox: And when I joined, all that there was was Gopher. And very quickly the—

Hsu: The web?

Cox: The web ~~became known~~ [emerged shortly thereafter] and I jumped on top of ~~that~~ [using it for teaching] with both feet. The whole idea was, "How can I use internet to support teaching?"

Hsu: Okay. And so was that, were you working on that the whole time you were at George Mason?

Cox: Pretty much.

Hsu: One of the really interesting papers on your website that I ran across was, you have a page called Social Construction of Reality. Where did that come from?

Cox: That was PSOL['s raison e'etre.]

Hsu: PSOL?

Cox: The department… The Program on Social and Organizational Learning. That's all about social construction of reality. And—

Hsu: Oh. So that program was very into that.

Cox: Right.

Hsu: So it's more like, the page is more like a syllabus? Is that—Because it lists a whole bunch of different references.

Cox: Yeah. I honestly don't remember the page very well.

Hsu: <laughs>

Cox: I know you're right. I know it's there. It's just I don't know what I poured into that.

Hsu: Okay.

Cox: Sorry.

<laughter>

Hsu: Okay. What did you think of the idea of social construction of reality?

Cox: Initially, I came from [the hard sciences which held] a much harder-edged view of the world where reality is what reality is. And only way to view it is through the scientists' lens. But I eventually came around to view social construction as being, you know, a better view of the world.

Hsu: Oh. Why is that?

Cox: It starts with that claim before, that science is the better view. Science is a stronger view, the objectivist view of the world. And then you realize, "What is science, if not a construct of community?"

Hsu: Right.

Cox: So… And then you go on from that and realize there's nothing other than community that's creating your understanding of the universe.

Hsu: Right. Did you get that from reading Thomas Kuhn or…

Cox: No. Certainly not Kuhn. But he was a part of it. Who [are the] main authors there? I'm articulating arguments that came from other people in that department more than things that I was personally invested in. But…

Hsu: Did you read Peter Berger and Thomas Luckmann's book, *The Social Construction of Reality*?

Cox: No. I never read it directly.

Hsu: Okay.

<laughter>

Hsu: But I guess it was in the air.

Cox: Yeah.

Hsu: What about the literature in Sociology of Scientific Knowledge or the Social Construction of Technology?

Cox: Yeah.

Hsu: Or Science, Technology and Society? Those fields?

Cox: Certainly.

Hsu: Okay. <laughs>

Cox: Yeah.

Hsu: Disclaimer. My Ph.D. happens to be in Science and Technology Studies.

Cox: Ah.

Hsu: And my dissertation advisor was Trevor Pinch, who was one of the founders of Social Construction of Technology. So <laughs> that's—

Cox: <laughs>

Hsu: —where my interest is coming from. <laughs>

Cox: I see.

Hsu: Kind of. You know. <laughs>

Cox: Yeah. Wish I could say I have a greater depth of knowledge about that than I actually do. This was more the sea I swam in than something that I contributed to.

Hsu: Right. But the ideas were—you were open to the ideas. They were attractive to you?

Cox: Yeah, reluctantly open.

Hsu: Right.

Cox: It took me some time to come around.

Hsu: Right. Was that difficult?

Cox: Oh, it took some time. So that probably means it was difficult, but… But I came around.

Hsu: Okay. <laughs> How much did you grapple with maybe the relativistic or post-modernist sort of ideas in some of these constructivist views? Because, I mean, I know coming—

Cox: Well, it was very much the environment that I was in. Again, what I grappled with personally was how to do teaching over the internet.

Hsu: Oh, okay.

Cox: That was where my personal energy was. But this whole idea of social construction in post-modern approach was very much the environment that I dealt with every day.

Hsu: Right. I see.

Cox: I just wasn't expected to contribute to it.

Hsu: Right. So what sorts of things did you produce in terms of teaching over the internet at George Mason?

Cox: Well, it all started with this whole view of, "Why are all of you," speaking of students, "sitting in this room? Is this room serving your needs or is there a better way?"

Hsu: Oh.

Cox: Why do we—parking was always a huge hassle. Why did we put up with all that hassle? Why did you take time off from work to come to this particular room when this new thing called the internet is claiming to be able to reach out and deliver education to you where ever you are? It all started from that line of questioning. As soon as we had something better than Gopher, the early browsers, I began ~~to see~~ [searching for] a way to actually deliver education at a distance. Long distance. Overseas and interstate. And just became very energized with pushing that to the limit. Where I ended up with was a Perl-based environment that would basically lead a student from nothing to being able to build websites, which was the goal I set for the course, without much ability to stray. ~~They didn't have much ability~~ [I didn't leave room] to not do the work, to skip the work, because the Perl program was constantly keeping them on the straight and narrow path. So that most—I felt like I could [pretty much] guarantee learning that way. And the results pretty much proved it. I could pretty much guarantee anybody who kept up with the demands of ~~those Perl scripts~~ [the programs] would get an A.

Hsu: Okay.

<laughter>

Cox: You know. It seemed very successful, except in the sense it was too expensive. The load I was carrying, supporting all of that, writing all that code to support that, was not sustainable in the long run.

Hsu: I see. So you were using that course to teach web programming or, like, creating a website?

Cox: Yeah. The technical term for it was building a website. What I was really after was how to teach quality websites. How to appreciate quality and deliver.

Hsu: Right.

Cox: You know, beyond the, ~~what was really required~~. [technical ability to actually build a website.]

Hsu: Right. Would that approach have worked for other subjects like physics or history or…

Cox: I don’t see why not.

Hsu: Would it have taught a sort of very narrow subset of the material or, I mean, because you said it limited exploration.

Cox: No, subject to ~~the~~ [my own] workload requirement of writing and debugging a whole Perl program every week.

Hsu: Oh, okay. <laughs>

Cox: Right. I don’t feel like it—

Hsu: Just <inaudible>.

Cox: It scales since there’s no 20-person limit on what one professor can do. I mean, there were semesters I was carrying like a hundred students. Internationally. Because these Perl programs did most of the work. But to scale it across the faculty for a whole curriculum, I’m not sure that’s doable.

Hsu: Right, I see.

Cox: But it sure worked well for what I was doing.

Hsu: <laughs> Did that work? I mean, was that an early version of what you now have called MOOCs, these massively online courses?

Cox: That was very much coming into fashion at that time, but it's not exactly, no. The whole idea was to create this tight interactive loop, using Perl to create a tight interactive connection between each student, me and the other students.

Hsu: Okay.

Cox: And these programs would do things like create collaborative tasks that had to be done in collaboration with other people on their team. And manage that collaboration so there could be no shirking.

Hsu: Right.

Cox: You know, very difficult programming that led to a very good learning experience.

Hsu: Right.

Cox: Because I remember <laughs> one of the tasks, once I taught them the technicalities of building websites, one of the tasks beyond that is, form up into teams and as a team choose a project that makes the world a better place.

<laughter>

Cox: And bring back a customer that confirms that you've done precisely that. So people were doing everything. They would automate pet shelters. You know, build websites to get dogs adopted. One of them built a website for Shell Oil Company.

Hsu: <laughs>

Cox: All over the map.

Hsu: Okay.

<laughter>

Hsu: That's interesting. I don't know, how does this Shell Oil Company website make the world a better place? <laughs> That's debatable.

Cox: That was <laughs> their challenge.

Hsu: <laughs>

Cox: They didn't have to explain it to me. [Just get their client to confirm that they did.]

Hsu: <laughs> Okay. And so you were teaching this over the web, so how was, like, were they running the program in the browser? Like, were they writing programs in the browser? How did that work?

Cox: Well, the browser supports forms.

Hsu: Okay.

Cox: So when they sign in to do their work they get a webpage that isn't just a blank syllabus that tells you generally where things are going to go in the future. It says, the login page tells them, exactly the next step they need to take to succeed in the course. This week. So they click "Okay," and it gives them some information and then a form at the end checks their understanding. And if they have to interact with other students, well, here are the links to the other students on your team. Here's what they thought about what the team should work on. In forms. You know, every one of them filled out forms and the form, everyone's form is then filled out by everybody else on the team. So you just use that to lead them down the learning path.

Hsu: Okay. So let's see. You were at George Mason for how long?

Cox: Gosh, five.

Hsu: Five years.

Cox: Five, maybe more. Five, maybe six.

Hsu: Hm. And why did you decide to leave?

Cox: Startup company.

Hsu: Okay. Started your own company or…

Cox: Yeah. This was a superdistribution company.

Hsu: Oh.

Cox: See if I could make a go of that.

Hsu: Right. And how long were you doing that?

Cox: About a year. This was right when the bottom fell out of the internet.

Hsu: Ah, okay.

Cox: Internet boom and it didn't make it.

Hsu: Right. Okay. <laughs> So then what did you do after that?

Cox: Ooh. After Stepstone. After superdistribution was… I think that's when I got into government consulting. Seems like I'm maybe missing a step, but that's pretty close.

Hsu: And what drew you to government consulting?

Cox: Just [largely because] that's what available in the area.

Hsu: Oh. Because you're in the Virginia—

Cox: Yeah. Not for any better reason than that. It's just what's available here.

Hsu: Right. So it seems like it's at this point that you start to sort of think about sort of these larger levels of granularity, SOA, service-oriented architecture, and you start using this new metaphor of mud bricks versus real bricks?

Cox: Yeah.

Hsu: So how did that come about? Was it work that you were doing for DoD or for other parts of the government that, you know, what led you to sort of think—because you hadn't thought of that before. How did you start thinking of it now?

Cox: Well, mostly because it was happening now. Before it wasn't happening. So it took years to mature and mobilize, and even then—so I don't have a better answer than that. ~~It just wasn't real~~ [SOA didn't become an option] until then.

Hsu: I see. So it's sort of this whole idea of selling services on servers on the cloud that sort of enables this, you know, the—

Cox: The [large-scale] component idea. [Appliance-level components, not just gate-, block- and chip-level ones.]

Hsu: The component to be marketed, to be sold for money now.

Cox: Right.

Hsu: As a service and not as a, "I'm giving you the thing."

Cox: Yeah. Making components successful, basically. So you could be succeeding on the cloud. The thing I never understood is why SOA went sour.

Hsu: I see.

Cox: I never understood that but it—

Hsu: What happened to it?

Cox: I think people just got fed up with the complexity of ~~doing it~~ the standards ~~for supporting~~ [that grew up around] SOA. But I'm speculating, because I don't have a good feeling for why people [seem to have] lost faith in that approach.

Hsu: How is that approach different from what people are doing now on the cloud?

Cox: What they're doing now is SOA without the standards. And that feeling of overbearing, overwhelming complexity of the standards.

Hsu: So they're just doing it in their own proprietary way?

Cox: Their own non-proprietary [way. Mainly open source.]

Hsu: Oh, their own open source way, ok. Rather than have some sort of get together and a committee and decide to create a standard, they're just doing their own thing.

Cox: I think what drove all the [over-bearing] standards with SOA was the government. And that drive just got out of control, I think. But I can't deny that people totally lost patience with it. I wish I had better facts.

Hsu: Is it important, do you think, to have standards? To have these sorts of international or national standards rather than the open source approach?

Cox: Well, it's—all I can do is point to the hardware industry, where, yes, it's very important. Because that's what's makes components work there. So now is it important for software? Well, my habit is when I'm looking for insight in software, I'll look to hardware for guidance. So based on that, yes, they're very important. But this is [neither science or engineering but a] craft based [mud-house] enterprise that is prone to inventing its own rules that—regardless of any other point in human ~~history~~ [experience]. So those lessons from the hardware industry [just] don't seem to have as much value to other people as they do for me.

Hsu: I see. Because of the immateriality of software, because it's infinitely copyable?

Cox: Yeah, and there's nothing to hang onto like, how did steel work last week. Can I build a table that shows how to harden it?

Hsu: But if that's true about software, if it's so immaterial, why do you keep going back to hardware to draw lessons from, if there's a fundamental difference?

Cox: This is my nature. I look for places I feel like I can get insight, and that's where I turn.

Hsu: I see. So you mentioned that the government was involved in trying to standardize SOA. What's your view of the relationship between the government and the market?

Cox: It used to be far more important than it is now. Today’s industry is pretty much going its own way regardless of what the government thinks.

Hsu: So this is just in terms of technology standards, you mean?

Cox: Right. The government once believed that it had more influence than it proves to have, but I'm speculating. I don't know.

Hsu: Right. And how much do you feel like—just not in terms of technology, but in terms of the economy, how much involvement do you think the government should have in the market?

Cox: As little as possible. <laughs> It's terrible at it.

Hsu: So are you more of a libertarian?

Cox: I suppose so, yeah.

Hsu: You're more of a small government versus big government type. Does that conflict with the fact that you sometimes, you were doing contracting for the government?

Cox: I suppose it does. I'm astonished at government's waste in doing things, but also would be very surprised, if they can change. There's room for conflict there.

Hsu: So you see a tension there.

Cox: Sure.

Hsu: OK. So you've also talked about, was it Java Business Integration, Software Component Architecture and something called OSGI. So what are these things? How are they making programming better?

Cox: Well, let me start with OSGI. That's basically an attempt to cure the complexity that comes from combining layers of software that are independently shifting, going through their own version changes. It can put you at what's called jar hell.

Hsu: Jar hell? Okay.

Cox: Where something down in that pile of shifting goo changes and breaks everything that depends on it. Well, it prevent—it's more involved than that, actually, but OSGI's an attempt to fix that by allowing you to build components where the versioning problem is nailed down and at least known if not resolved. That's—OSGI was the name of the project that invented the approach. That approach was eventually adopted and is being adopted into the Java core in a project whose name I now forget. It's not quite available yet, but it's hoped for in the next release. I wish I could think of the name. But OSGI, you can think of it as a way to solve the technical complications of building a component based system [when every layer below you is shifting.] Now JBI, Java Business Integration, I'm not sure that's still alive because I haven't heard about it for years. But I was interested in that as a way of using components that can be snapped together with a very simple pipeline kind of a metaphor. You can think of it as like copper pipes in a plumbing system.

Hsu: Oh, right. That's another metaphor. Plumbing, you used that metaphor in your papers, also.

Cox: Right. Yeah, but it's stream based programming, basically, where the components are stream processors, filters and water heaters and, you know, things of that nature. JBI was one of the things I tried, to support that approach before it vanished. I think it may have died with SOA, maybe [is] what happened to it. And you mentioned another one.

Hsu: SCA, Software Component Architecture.

Cox: Yes, that was a part of the JBI approach.

Hsu: OK. It seemed that a lot of this level of—I think you're identifying at this point that the problem is creating trust for components. So rather than selling bits, we need to sell trust in the components. And that's one of the problems that software ICs didn't succeed with, because you couldn't necessarily trust the components.

Cox: Well, it's hard to be that simple about it. Consider what did it take to go from, let's say, mud and sand houses to brick houses? Well, it's a whole lot of stuff that took millennia to do. Let's say it started with the Romans, who had to invent standards, then we had to invent standards bodies, and we had to invent testing labs. We had to find ways to incentivize all of those very costly kinds of efforts. And it took millennia to do it with simple stuff like steel and bricks. The end result of all of those standards and all of tests is trust. But trust is what comes out of the pipe. What goes into it is millennia of trying things. So I think that's what I was trying to articulate.

Hsu: So are you saying that we need a millennium of experience in software, before we can fully trust software components?

Cox: I think so and probably more than a millennia, because we carry such a liability in incentivizing the production of software.

Hsu: Because the market mechanisms aren't there or are not the right ones?

Cox: Yeah, because it's so hard to—without scarcity, what will replace it? Well, maybe the answer is cloud computing. Well, that's barely here today. Maybe a millennia starts today and goes into the future, but I'm not an optimist, because it's such a liability, economics-wise, that we're carrying with us.

Hsu: So you don't think that trust can be created. You think that trust has to just emerge naturally?

Cox: Evolve is a better word for it.

Hsu: Evolve, right.

Cox: Yeah, I think so, because I don't know about you, but the idea of putting a computer in charge of driving a car is just—I can't express the level of idiocy that represents to me. I don't trust it. I've never met a computer I trust more than—they don't deserve trust. <laughs>

Hsu: <laughs> Well, so you're pretty skeptical of AI, then? Or machine learning?

Cox: Don't get me started on AI. [Even though that’s exactly what I’m expected to do these days.]

Hsu: You wouldn't put your life in its hands.

Cox: No. And remember this is what I do for a living.

Hsu: Right. <laughs>

Cox: You know, machine learning and…

Hsu: Oh, is that what you're working on now, is machine learning?

Cox: Well, cyber computing, machine learning for cyber protection.

Hsu: For security.

Cox: Yeah, but as far as trusting it, I would never use that word. You know finally the reason we talked about, it's complicated. Steel, I would trust. Mud bricks and computer programs, I don't. Steel, I trust, because there's standards, because it's tested, because it's been experienced over millennia. None of those are true of software.

Hsu: I see. But if we did have those standards and a millennium of experience, then maybe we would get to that point where we trust the software.

Cox: That's a maybe.

Hsu: That's a maybe. <laughs>

Cox: <laughs>

Hsu: Interesting. Since we talked about AI, so you've also said that you don't get along with LISP or Haskell or other functional languages. Is that still true?

Cox: Oh, gosh. That doesn't sound like something I'd say literally, but maybe figuratively. Haskell, I gave it an honest try and I just couldn't. I fail to understand the syntax. The idea of it is, yeah, it's an important idea, stream processing, basically, or functional programming. But I can do that in a simple language like Java without putting up with the pain of learning a totally new syntax. So I tried, but I failed to understand it.

Hsu: So it's just mostly the syntax and the way of thinking is too alien?

Cox: And the way—they don't seem to—they, the people who promoted Haskell, don't seem to be able to put three sentences together without a word like “monads,” which doesn't help [newcomers] to climb the [learning] curve.

Hsu: Did you have similar issues with LISP?

Cox: I don't feel like I did. LISP just got parentheses in the wrong place, but that you can cope with.

<laughter>

Hsu: Right. So what do you think of what Apple has done with Objective-C in the last couple of decades?

Cox: Oh, they've done miracles. It's, frankly, been wonderful to watch what they're doing.

Hsu: Have you ever written code using Apple's Objective-C 2.0?

Cox: One. ~~I have one project, an iPhone project.~~ [I did do an iPhone application.] I was impressed [with the tooling.]

Hsu: What do you think of Apple's new Swift language?

Cox: To tell you the truth, I've come around over time to the view that what this world needs isn't more languages. So every time something new comes out, I don't rush to learn it. That was my reaction to Swift. I just didn't learn it. Because I've got all the work I can do right now with the language that I do know, Java. And that's where I've concentrated. Sorry.

Hsu: OK. Well, so you mentioned that right now you're doing stuff in machine learning and security. And so could you elaborate on what your current work is?

Cox: Well, it's basically to detect—getting as early a warning as possible of attacks that are underway.

Hsu: Like hacking?

Cox: Yeah. Many of our clients are government clients and ~~they're exposed and, you know, attacks unfold~~. [They're very concerned with that.] It takes months to complete an attack, from doing the reconnaissance, to ~~collecting, of~~ learning how a network is put together ~~from outside~~ to actually carrying out some attack ~~and exporting~~ [to] exfiltrating the results. It can take months to complete that chain and we're trying to break into that chain early and raise an alert of exactly where in the attack process things are right now and here's how to head off the next steps. There's a desire to apply machine learning to that problem. Many of the ideas there are yet to be proven. There are people who've tried it, but I don't know that anybody claims to have successfully explored that approach. So that's part of our goal.

Hsu: And this is a company that sometimes contracts with the government?

Cox: Yeah.

Hsu: Okay. And so, was this a natural outgrowth of your previous government work?

Cox: I would say it's—yeah, but also back to, remember that early work in machine learning, neural networks at University of Chicago?

Hsu: Oh, right. That's coming back. Exactly.

Cox: That's coming back. And I keep looking for a way to get back in touch with that. I haven't found it yet, but I keep hoping that I can head back in that direction. Maybe someday.

Hsu: So the machine learning is a way to get back to that. So do you consider yourself more of an academic or a industry person and do you consider yourself more of a social scientist or an economist than a computer scientist or a software engineer?

Cox: Well, no to the last two, because those don't exist.

Hsu: <laughs>

Cox: Most of my career, I've called myself a software developer. Recently, increasingly, I'm getting more and more aligned with data science. [My latest job title is] data scientist.

Hsu: Analytics. Big data.

Cox: But I still do software development to do that. Certainly not academic. I didn't enjoy academia at all.

Hsu: A lot of your career has been about higher levels of encapsulation—components, larger granularity. Is that sort of, most of your career, you would say, has been focused on that? And is that an accurate statement?

Cox: I would say it's been motivated by a desire for that, certainly. I don't feel I've been totally successful at that, because of the business model issues. But where those issues are resolvable, like for cloud computing and for Apple's components niche, I feel some success from those.

Hsu: Are there problems that components don't solve, in software?

Cox: I could give a flip answer and say they don't cure cancer.

Hsu: <laughs> In software, then.

Cox: Well, let me just think out loud for a second. In hardware are there problems that components don't solve? Well, probably, yeah. I mean nothing solves everything. But I'm not being responsive. I'm beginning to see a real problem, see that the problem that needs to be most solved is not so much productivity as complexity. And components are a real help with that, but I'm not sure they solve that [completely either.] There are always unsolved things at the edges of the things that you do solve. And the complexity problem, I can't imagine that that can ever be solved.

Hsu: I see. Well, where do you think programming languages specifically or components or the software industry in general is going or should be going?

Cox: Oh, Lord. I almost wish they wouldn't.

Hsu: There wasn't a software industry?

Cox: No, that they wouldn't be—I'm not sure the answer to anything is more languages. So I meant to say I wish the industry wouldn't [add] even ~~do it~~ [more languages to the pile it has already.]

Hsu: Do more languages.

Cox: Do more languages. It's like, more [computer] languages is like more human languages. Why would you want more human languages? If you've got the old British sets of weights and measures, is adding the metric system not just making things worse? Adding languages, adding systems, adding incompatible standards. I don't see it as a solution to anything, but as a way of making the problem harder.

Hsu: But wouldn't you say that, I mean, like human languages, social needs change and so the new languages better address those needs than the old ones?

Cox: Yeah, you're probably right. But I was thinking about different languages, French and German and Latin and all of that. Is that an advantage or a liability?

Hsu: The Tower of Babel, you mean.

Cox: Exactly.

Hsu: The incompatibility between multiple different languages.

Cox: But, you know, I know I'm tilting at windmills here. This industry is going to spew languages as long as it's got a life to spew.

Hsu: What about software in general? Where do you think it's going or should be going?

Cox: There's a fellow at HP Labs, whose name I've forgotten. But he's sort of leading the charge for something called capability-based architectures. Do you know what that means?

Hsu: I've heard of it, but I'm not familiar with it.

Cox: Well, Ken Hamer-Hodges, I mentioned him earlier, put me onto this idea. He was involved very much in building capability machines at Plexus in Britain before he came over here. And the fellow at HP whose name I've forgotten takes a game of solitaire as an example of an extremely powerful program. It's a program that can write checks, [it's got all the power it needs to withdraw money from your bank account,] it can write defaming letters to your boss. That program can do anything you can do, because solitaire has exactly the same privileges that you have on your computer. It can do literally anything that you can do. So it's a very powerful program. And that notion of power that you, the computer owner, delegate to every program that you have on your computer is the foundation of how computers work [today.] Well, that to me seems like not the kind of foundation you want to trust. And that's basically the point of all this and the point of this desire to switch to a capability-based system. The problem goes from there to the—from the computer—trusting the computer to trusting the network that joins computers together, which seems even more untrustworthy. So where do I think computer software is heading? It's heading to a recognition that we're building on an untrustable foundation and, ~~hopefully~~ [I hope that leads to] the determination to [finally] do something about it.

Hsu: I see. So the capabilities-based system, it doesn't solve the problem of trust, it just makes it worse?

Cox: No, ~~that's~~ [it provides] a technical solution to the [delegating too much power to computer applications] problem [by specifying what privileges they should have before allowing them control of the computer.]

Hsu: To the trust problem.

Cox: Right.

Hsu: Is by imbuing these programs with more power?

Cox: ~~With~~ [Providing] limitations to the power [that we grant them by default today.]

Hsu: Oh, with limitations to their power. Okay. You're building in limitations to their power and, therefore, that allows you to trust it more. Okay, I see. You've mentioned Ken Hamer-Hodges a lot. What's been your relationship with him over the years?

Cox: He's just a very good friend. He worked—he was at ITT, head of the research, the development labs there. Then he joined Stepstone and we've kept in touch over the years.

Hsu: My last question. What advice would you give to a young person interested in computers today?

Cox: Oh, Lord. I've got great hopes for the Spark environment.

Hsu: Spark?

Cox: Right. And I think I would advise a youngster to get started down that trail. It's a very painful road. Spark is still pretty ~~awful~~ [dreadful] right now [but it has huge promise.]

Hsu: Is that right? Is that an acronym?

Cox: No, it's…

Hsu: S-P-A-R-K?

Cox: S-P-A-R-K. It's basically ~~after~~ [a layer on top of] Hadoop. You know what Hadoop is? They're real big in the big data world. They are the foundation for big data work.

Hsu: Okay, so it's a machine learning data analytics kind of a thing?

Cox: That ~~goes~~ [is a layer that runs] on top of SPARK, but yes. SPARK is.

Hsu: SPARK is the platform on which those things are based.

Cox: Yes. So the foundation down at the bottom is Hadoop, which is a storage and remote execution environment that works in a distributed network. Then Spark runs on top of that to support a streaming approach to computation. Remember the pipes and filters I was talking about earlier? They recur in that space. And then there are machine learning libraries that go run on top of Spark. So it's a very complicated and difficult environment [right now,] but I think that's where I would point youngsters since Big Data's not getting any smaller. There'll be a job there for them.

Hsu: Right. So AI, not necessarily, but…

Cox: It's not my favorite, no. It was basically my frustration with AI that got me to the neural network approach to computing. I was trying to articulate why I felt so uncomfortable about AI by doing that work.

Hsu: Right. That's classic rule-based AI. But, I mean, current machine learning is not the same thing as—. That's more neural nets…

Cox: Oh, no. I view that as vindication of what I thought back in graduate school.

Hsu: Oh, OK. So machine learning using neural nets is now the new form of AI in a way.

Cox: that label has been pushed in that direction.

Hsu: But you're not fully comfortable with that labelling?

Cox: Well, it's just rule-based. What I didn't like was this idea of [a]rule-based [approach] as ~~leaving~~ [offering] any new knowledge about how biological systems work, because they just don't do it that way.

Hsu: Right. But the neural nets approach, that's the way to go.

Cox: So I believe.

Hsu: And you think that sort of these sort of autonomous learning machines will, do you think they'll eventually take over the world or do you think...?

Cox: Well, no, I don't.

Hsu: So they have inherent limitations? How do we control them when we don't even know exactly what they're doing?

Cox: Pull the plug. [Really. I've never seen a computer the slightest bit smarter than a rock, regardless of what the marketing hype is trying to make us think they can be trusted to do. Control my car? Control my home? Control nuclear power plants? No thanks.]

Hsu: <laughs> What if they're just out there on the network self-replicating?

Cox: Sorry. That just doesn't compute for me. The idea that something as dumb as a brick can be that kind of a threat. It just doesn't compute to me. I mean, so you pull the plug and kill it. It's like...

Hsu: So you don't agree with the Bill Gateses or the Elon Musks who are worrying about smart AI capability?

Cox: Not at all [so long as computers are smart as bricks and so utterly exposed to malicious actors.] Not going to be a problem.

Hsu: Well, that's it for my questions. Thank you very much for giving us your time and inviting us to your lovely home.

Cox: Thank you very much.

END OF THE INTERVIEW