Department: Interviews
Editor: Dag Spicer, dspicer@computerhistory.org

Oral History of Alan Cooper

Hansen Hsu
Petaluma, CA

Hsu: The date is March 13th, 2017, and I'm Hansen Hsu, curator, Center for Software History, and today, we are here with Alan Cooper.

Cooper: I was born in San Francisco, California. I like to say I was born in Frisco, just to drive people crazy, in 1952, and right smack in the middle of the century. And I lived there for a few years and my mom wanted to move to Marin County. She wanted to move to the suburbs. My dad, who worked as an electrician in the city was kind of content, I think, with the city life, but my mom had this vision of, you know, the postage stamp lawn and the little house. And so when I was about four years old, we moved to Marin, and that's—I grew up in a little, tiny town called Kentfield, really a bedroom community.

Hsu: Can you tell us a bit more about your childhood?

Cooper: Well, I'm sure I was just insufferable as a child, because I'm very sort of self-centered and self-indulgent, and critical, kind of a creative design type. And those are all the things that get you punched out in the schoolyard, you know. And so I was not really a happy kid, and I struggled for a long, long time, and it was—in many ways, it was marijuana that rescued me, because number one, it was a different way of thinking, but it was a counterculture that I could participate in, you know, so it became my peer group, and I had a peer group for the first time. I wasn't actually interested that much in pot, and I stopped using it pretty quickly, but I had a group, we had long hair, and, you know, I had an attitude, and I did psychedelic light shows, and I hung out with musicians and artists, and it gave a center to my life. Everyone needs a center.

Hsu: So what did your mother and father do? What were their occupations?

Cooper: Well, my mother was a homemaker, you know. That was back in the day, and my father was an electrician. He loved that. He loved the construction trades, all of them. Even though electricity was his thing, he loved to use his hands to make things. He made stuff out of wood, and metal, and he knew all the trades, and he could plumb, and tile, and everything. And he became after a while, while I was still a teenager, he became an inspector, an electrical inspector. He worked for San Francisco, and so he kind of graduated to a more white collar job. And he also taught. He

Digital Object Identifier 10.1109/MAHC.2020.3033744
Date of current version 15 November 2020.

taught apprentice electricians at a vocational technology school in San Francisco, and he was generous enough to actually let me attend some of his classes. So as a 12 year old kid, I'd sit in the back of the class with a bunch of 20 year old guys. It was all men at the time, of course, and he would teach electrical theory, and I followed as best I could. But that was a real treat for me to watch my dad standing in front of the class.

Hsu: Do you have any siblings?

Cooper: I have two sisters. You know, my parents, they were each the youngest of their families, and I'm the youngest of theirs, and so there's a long distance. Both my folks have been passed for many years, and my sisters are moved away, and we're not really close, so my family really is my wife's family. I married into a much more amendable family to me, and as I mentioned, Dave Carlick already was an important part of that, and so I've learned a lot from my wife's family. They're from Paducah, Kentucky, so they have this sort of southern gentility that was foreign to me. I found it endlessly fascinating. They're very gentlemanly, and courtly, and so they've been my family.

Hsu: Okay, thanks. So obviously you're very into art as a, you know, your whole life. How did you get into that, and did you have any other hobbies besides that?

Cooper: I have a million hobbies. For one thing, design is not art.

Hsu: Mm-hm.

Cooper: And when I was in school, I went to the College of Marin, a local community college, a very good community college, and so I like to say I spent five of the best years of my life at the two-year college. I studied, because I wanted to get into architecture, I was trying to—I was doing essentially prearchitecture at a community college so that I could get into Berkeley, and prearchitecture at a community college meant art and design. So I studied art and design, and art is free expression. Design is a much more focused kind of critique, the sort of a studio way of doing things, and I learned a lot from that. I had pretentions as a youth to be an artist. I, first I thought I was going to be a great photographer. Then, I thought I was going to be a great painter. I thought I was going to be a great printmaker, and it became very clear to me that I was not, that I was not an artist. I was a craftsman.

I loved the tools and the trappings of art. I had boxes of brushes, and pencils, and paints, and I loved all that stuff, and I knew more about it than anybody else, but I was not an artist, because an artist has a story to say, to tell, and I didn't. And I realized that there was a different role for me, and that's the thing about design is design is problem solving. A lot of people call it design thinking today. That's not a distinction that I make, but it's a convenient term that's gone out into the wild. But to me design—I call it "Alexandrian design" after Christopher Alexander, who happens to be a professor of architecture, or formerly professor of architecture at Cal Berkeley. He talked about design as a problem solving discipline. Design is the synthesis of a form to fit a context, and when I read his work as a boy, I mean, I was probably 14 when I read his book, "Notes on the Synthesis of Form." I found it in my high school library, and it rocked my world, because I had been reading books on designing houses. I read Christopher Alexander's work, and it was a theory of design. It was a way to think about what design is, and it blew my mind.

I love hobbies. To me in many ways programming and design are hobbies, you know? Some of the hobbies I've built companies around and some of the hobbies were just fun. So, when I was a kid, I liked to swim, and ride my bike, and with Gary as a buddy, bicycling was an interesting thing. I've always been interested in trains, and model railroads, and model railroading is a really interesting hobby because there are so many different aspects of it. A lot of people think it's about building toy trains, but the part of model railroading that has become so fascinating to me is the historical research, so I'd go on these trips to middle America, and I'd go to the bad parts of town, and take pictures of old shuttered factories, and stuff, and go into historical libraries and research, you know, industries that were thriving in the 1950s, things like that.

Hsu: Were you into science and math as a kid?

Cooper: I was into science. I liked science, but not math. I'm a visual thinker and a visual learner. And the way math, I don't know how it's taught today, but the way it was taught in the 1960s was antithetical to the way I learned. I had

a saying when I was into model rocketry... which is that model rockets might get a kid interested in mathematics, but mathematics won't get a kid interested in rocketry. And that's kind of my philosophy: mathematics is the burden if you want to make cool things. And, you know, beyond multiplication, my eyes glaze over, but I'm really interested in the applied stuff. I mean, Adam Savage is a great hero of mine, 'cause he says, "Kids," you know, "the only difference between science and screwing around is writing it down." And I love that notion.

Eventually, I started taking classes at the College of Marin. As soon as I finished the first mandatory introductory course, I threw myself into the data processing curriculum, and I got a work study job as a computer operator inside the computer lab. The college's computer, they had an IBM System/370 also, and it not only was used for teaching, but they also administered the school on it. So as a student operator, they didn't let me run a lot of the big jobs that ran the school or ran the district, but they let me run all the student jobs. So I worked inside the actual computer room, and the other students would hand me a deck of cards through a window. I'd run the deck of cards through and then I'd take their green bar print out results and rubber band it together and toss it in the out basket. But it meant that I was inside the air-conditioned room with the big computer, and they were outside. So I got all the computer time I wanted, so I wrote bigger and bigger programs and did more and more stuff. But that was, you know, a student job, a work study job.

There was also a timesharing culture. The minicomputer timeshare thing was growing, and I spent some time at the Lawrence Hall of Science in Berkeley, where they had timeshared computers. You could get time on them. I don't remember if they were free or dirt cheap. And there was also, there were some guys down in Silicon Valley. It was a long drive for me, but there was the People's Computer Company. But also right about that time is when the Altair 8800 came out, and then the IMSAI 8080, which was my first computer, and I went to a buddy of mine, the guy who had said, "You're over qualified. Go get a job." And I said, "Look. We could buy a computer with money that we, actually, have."

And he said, "Wouldn't it be great if we could find some guy who was about to buy a turnkey minicomputer accounting system? And, you know, instead of spending $30,000 on a [DEC] PDP-8 or something with an accounting system, we could sell them a microcomputer system? And what we could do is we could sell it to them—," he outlined the business model. He said, "We'd sell the turnkey system to them, but then tell them that it's going to be three months before we're ready. And then, in the three months, we'd write a business accounting system, and then we'd give them the microcomputer and this business accounting system, and then we could keep doing that." And I said, "You know, I was at a party just the other night and I was talking with a guy who was an accountant, and he was thinking about buying a system." So, we went and we talked to the guy. He promised us the money, we promised him the software, and we did eventually sell him a system. But he never did front us the money.

Hsu: And that's the IMSAI?

Cooper: Yeah, that's the IMSAI 8080.

Hsu: Wow. Let me take a little step back. So how important was the counterculture in exposing you to the early microcomputer hobbyist scene?

Cooper: Well, having your own computer was pretty darn countercultural. And so it wasn't that the counterculture had computers, as much as it was that countercultural people did things that other people didn't do, and one of those things was have your own computer and figure out something to do with it or start a company around it. And so there were a remarkably large number of people who picked their own path in the early days, and there were self-help communities like EST, and the whole IMSAI computer company came out of EST.

Hsu: Were you reading Ted Nelson or Stewart Brand, people like that?

Cooper: Oh, yeah. Yeah, Stewart Brand, the Whole Earth Catalog was the bible of that generation. It was a printed Internet is really what it was. I mean, a World Wide Web, I should say. Because he called it "Access to Tools." It was a catalog of all the stuff that creative, outside the box, thinking kind of people would want, and it was a very important publication. And Ted

Nelson's "Computer Lib," yeah, he was a radical thinker.

Hsu: Let's talk about your first company, Structured Systems Group. Were there any other personal computer software companies, at the time? Or were you one of the first?

Cooper: We were definitely one of the first. There were personal computer software companies. There were three, and Gary Kildall really is the guy who, kind of, said, "Hey, look, there's three companies. There's my company—" Gary Kildall's company, Digital Research, that did operating systems. They did an operating system called CP/M. It was the first. And there was Bill Gates, who had the first language that was commercially viable, which was Microsoft BASIC. And then, there was my company, Structured Systems Group, which had business application software. And that's how he [Kildall] articulated it. He said, "Really, this is the three points of the tripod."

We developed our own distribution network. It was, basically, a Rolodex filled with the names of little computer stores around the country, and we had developed a channel and it was very clear that if we could invent more software, we could drive more software through that channel. So it was a matter of trying to say, "What can we do?" And so I wrote a little program that managed names and addresses, and then, I wrote another little program that let you create, like, little comma separated value data files, and then reassemble them and print it out as a report. So it was a very primitive database manager kind of program

I started Structured Systems Group in 1976, which is when we discovered the microcomputer, came up with this idea of the turnkey [accounting] system, and built the General Ledger. I sold my interest in that company to my partner, in 1980. The company, we were pretty successful at the time. I mean, today, it would be down to a rounding error, the size of the company. But when we started that company, we would sit around and, you know, drink red wine and talk about how we'd be better than our bosses, at the big companies, at doing this, at running our own company. And I remember saying, "Yeah, I'm going to have my own company so I can get $50,000 a year," which was, like, the biggest number that I could think of. And a year later, I was making $50,000 a year. I was 26 years old and, you know, 10 years earlier, they told me I was going to be a ditch digger, and $50,000 year was twice what my dad made. I bought him a car. But people are weird, you know? And I felt like I was on this roller coaster. I kept thinking, "When are the grownups going to come in here and take this away from me?" And it became clear that what we needed to do was get some investment money, and I didn't even know what investment money was, but I knew that we needed help to take our company to the next step. But my buddy who was also, kind of, a counterculture guy, was—he was extremely happy at 50 thou a year. He bought himself a house in the suburbs and he said, "This is good. I don't want to go anywhere." And I said, "Let's go to the next step," and we started fighting over that. We started fighting over what's our destiny, and as people do, we started poking holes in each other's work, and it became untenable and something had to give. And being this young kid with no experience, no education. It had seemed very easy. I just did what was fun, and I'd built a successful company. What I did was I said to my partner, I said, "Just pay me my salary for the next 10 years, and you can take my half of the company," and I walked out.

That was in 1980. In 1981, he stopped paying. In 1982, he went out of business. There's a lesson there. You know, the wrong guy stayed and I learned that you need to watch your back. In my second software company, we did some pretty good stuff. I built a spreadsheet that was pretty darn good and was halfway done with a word processor that was also pretty ambitious. ... Well, in 1981, IBM released its PC, and there was an instantaneous consolidation in the microcomputer business, and the guy who financed me just started circling the drain. I had been negotiating with somebody to license my spreadsheet, and the guy who was financing me stepped in and forced the deal, because it was his money. I wasn't that happy with that deal, but what could I do? And I don't know what the timing was, but within a year, probably within six months, both, the guy, the microcomputer company, was investing in me, and the software publisher who was my one client for my spreadsheet, they were

both out of business, victims of the consolidation. I sat down one day and paid all the bills, paid off everybody I owed, quietly closed the doors to my second company, Access Software. And I learned that just because you're smart and hardworking doesn't mean you've got a success there. I learned more, I think, from failing than I did from succeeding at my first company. And it was at that point, that I took a job working for Gary Kildall at Digital Research, and he said, "Come and start the R&D department."

And what I had done at Access Software that was really interesting was I said, "I'm going to do things differently at Access Software." I hired a couple of guys who I really liked, who were good, strong programmers, and I said, "I want you guys to do all the coding." And I hired a woman, who was a technical writer, to do the documentation. And I said, "I'm going to specify every pixel and every keystroke, and I'm not going to write a single line of code." This was radical in this new microcomputer industry, and then, we also did user testing and it was, kind of, a semiblind user testing. We draped sheets over stuff in the office, and we'd bring people in and pay them 25 bucks, and they'd sit there and we'd give them tasks and we'd observe them. Again, this stuff was not done, and we had really good results. We really learned a lot, and I found that I was designing the interface, but I wasn't programming it, so I was detached from that. So when I went to Digital Research, I told Kildall that this is what I had done and I wanted to explore it, because, I said, "This is really promising." It's really good. I'm getting better results than if I just go head down and code, and he said, "Yeah, you can do that. So come on down to our R&D lab."

So I was the second guy there, and my wife joined Digital Research, too, in the marketing department and she had a much more successful time there than I did. Because, while Gary Kildall, the founder and owner of the company, told me what I could do, it turned out that they had a president who had been brought in by the venture capitalists, who looked at me and said, "No. I want to see you coding." <laughs> And I said, "No. You don't understand. I came here not to code." And he said, "Well, let's see what you do." And I just, kind of, went, you know, "I'm not going to do this." And so I wrote business plan after business plan for a product within the company, or I should say product plan. And each one, each plan, basically, said "I need a head count. I need a couple of software developers." And they never approved any of them, and so after about a year, I said, "I'm out of here," and I quit.

That was somewhere around late 1982, early 1983. And I went home, and I looked at this list of product plans that I had given them that they weren't interested in, and I said, "Any one of these I'll build them and we'll make a lot of money." So I took the top one off the list, which was a critical path project management program, and I said, "That's an interesting challenge." So I started to write it, and at the time, we had about three months' worth of money, my wife and I.

I had a spare bedroom in the house, and I just started coding, started writing this product, and it took about six months to build it to the point to where we finally sold it, and then we ran out of money. It was pretty horrendous. They began foreclosure proceedings on our house. PG&E came out and disconnected our electricity. I had to go out—my dad was an electrician, and I hotwired the house <laughs> so we could get the lights back on so I could run the computer. And I wrote, I think, a really good program. It was the critical path project management program that did tasks with interrelated dependencies. It was very cool stuff, and. . . then, I sold it to Computer Associates, and it was released as SuperProject. And I forget the numbers. It was, like, I don't know—they wrote us a check. What I remember is we had a check upon signing. So when we inked the contract, they gave me a quarter of the money or half the money, or something. And Sue and I went home and sat down and got out our checkbook and started writing checks to all the people we owed money to, and we spent $60,000 in an afternoon, because we had, basically, been living on nothing for three months. And then, we threw a party. <laughs> We called it the reboot party, and we pasted the walls with green bar paper of source code and we just celebrated. We invited all of our friends and we had a good time. And so we were back on our feet, and I built software like that, just going off and just thinking it up all by myself and then building it all by

myself, and then taking it out and showing it to software publishers. I did that two more times in the 1980s. I built three products, but the fourth one never saw the light of day, and the next one was the first serial communications program for Windows and I sold it to Will Hearst's company.

Hsu: Software Ventures?

Cooper: Software Ventures. And. . .

Hsu: And that was the Microphone II product?

Cooper: Yeah, and that was a—I mean, this is before the Internet. I should say, definitely before the World Wide Web, the Internet was an infant at that point. So communications were really serial at that point, so that was an interesting program. I was experimenting with a geographic information system that I was writing, that would do—that would display maps of different sizes and shapes and colors to indicate different values, you know, visual communications. And but what I really needed was a multitasking graphical user interface operating system. And such a thing didn't really exist, so I started writing my own, and it ran on top of CP/M, or DOS, I guess I should say. And that was an interesting problem. I mean, that was the way things were back then was if you didn't have it, you made it. And but it was a big project.

Hsu: What year was this?

Cooper: This was probably 1986?

Hsu: Okay. And so weren't there other graphical interfaces for DOS machines?

Cooper: No, no. And that's what this guy said, one day he said, "I want you to see what Microsoft is doing." And he took me to a technical pitch meeting somewhere in Silicon Valley, and there was Steve Ballmer up on the stage, and they were talking about Microsoft Windows. And they were saying that they were going to do really impressive stuff with this graphical multitasking system, and which did not impress me at all. Okay? 'Cause I had my own, and I wasn't that impressed with theirs. But there were some things that I couldn't do because I didn't have access to the deep guts of the operating system. And there were things that I wanted, which were interprocess communication, dynamic relocation, and dynamic load ability. This is the argument that I had for a year-and-a-half at Digital Research is, is all the systems guys there would come to me and say, "Alan, you're the applications guy, what is it you want?" And I'd say, "I want dynamic relocation." And they'd say, "Yeah, yeah, but why would you want that? What else do you want?" I go, "I want dynamic relocation." "Well. . .," "Yeah, yeah, no thanks." And they didn't understand it, is that without dynamic relocation, multitasking is useless. They saw it as a scientific experiment; and I saw it as a practical platform, and what you have to be able to do is as programs come in and go out of main memory, they—you have to be able to slide the remnants of the old ones down to the end, and make room for the new guys to come in. And they had the technology, they wouldn't build it. And it really frustrated me. And there was Steve Ballmer up on the stage saying, "Dynamic relocation and interprocess communication." And you know, dynamic loading of modules that could run and go out without shutting down the operating system. I said, "Okay, I'm sold!" It wasn't the sexy graphics that sold me. It wasn't the GUI that sold me, because I had my own GUI. What sold me was the dynamic relocation, which I couldn't do. So I went home and I put my little graphic front end, and multitasking dispatcher in the sock drawer, as my wife calls it, and started building software in Windows.

And the first thing you learned when you started working in Windows back then was what an incredible piece of shit it was. In particular, the thing that so was shitty about it was the shell, the "Finder," the face of it. It had this awful little program that was unigraphic and miserable. And it just was, just bad! And you know, Microsoft was doing whatever Microsoft was doing. They were building it or selling it or improving it or adding to it, or whatever. But the interface was just kind of the red-headed stepchild that nobody cared about. And so they might as well have had a neon sign saying, "Market Opportunity." And it just really intrigued me. So I started saying, "Okay, I'm going to build a shell." And so I started writing little programs that could be shells for Windows. But that's actually—that's a hard problem! You know, what would a shell be for Windows? It's an operating system that serves a lot of people, and it really—

Hsu: So you wanted to replace the Windows Explorer with your own?

Cooper: Yes. Yes. It was—their Explorer was just really bad. Everybody knew it was bad, even Microsoft knew it was bad, too, but they just didn't have the bandwidth or something, or the interest to make it any better. So a friend I had worked with at Digital Research had actually gotten a job in the sales team at Microsoft. And he asked me if I would come out with him on a sales call one day, 'cause I was—he knew I was a Windows developer, he was trying to sell Windows to these big IT Departments, and he was trying to convince the big IT Departments that there were actually guys writing for Windows. So I went along as Exhibit A, an actual tech guy. Because the jury was out at that point as to whether Windows was going to be a success or not.

And so where he took me was to the Bank of America Headquarters, way out in the East Bay. And I met with the Head of IT there. And my buddy at Microsoft is pitching the virtues of Windows, and I was going, "Yeah, Windows is wonderful." And this guy was saying, "Our issue here is that we've got people who are Senior Systems Analysts building systems. And we've got bank tellers who have a high school diploma, and they all have to use this system, this Windows system that you've got."

Plink! There it was.

Right there! I saw it! I saw that this is what the shell needed to be, I didn't want to write a shell for the expert, and I didn't want to write a shell for the beginner. I wanted to write a shell *construction set* so that the systems guys could build a shell for the expert, or they could build a shell for the intermediate analyst, or they could build a shell for the rank beginner—it would let them do what they needed to do. And so I was helping my buddy at Microsoft, and I went home and I started working on the shell construction set, which I code-named Ruby, no connection to the language. And it was—and it turned out to be really good. 'Cause it was—it had a little palette of controls, and you could click and drag-and-drop, and because of the dynamic relocation and the dynamic loading of these dynamic link libraries, DLLs, what Microsoft called them, you could build a shell on the fly that was pretty darn powerful, and you could wire them together and add little bits of very simple instructions. It was not the first use of dynamic linking. But it was the first kind of big useful use of it that really worked!

And when I had a prototype, that worked. I mean, it was really cool. You could take the blank screen, and you could drag-and-drop. You could create a little window and start putting little controls in it, and you'd push a button, and it would fire up a list box, and it would let you select a file, and then it would launch a program that would do something to it. It was simple shell stuff, but it was immensely powerful, because it was a visual programming language. And it had visual idioms in it. Like it had drag and drop, and you could drag an arrow from one control and the arrow would swing around and point to the control, and then when you pushed this control, it would affect that control. And this stuff didn't exist. The other thing I did, in order to have it all be visual, is I had to invent my own drag-and-drop protocols, because the drag-and-drop protocols didn't exist. And I had to invent sprite animation, because there were no sprite animation tools in Windows. I mean, it was really a primitive system.

And I—so just the same way I had shown SuperProject to a lot of people before they bought it, I went around showing Ruby to a lot of people, because there were a lot of publishers out there. And you know, and I showed it to Adobe and Lotus and all these guys. And they all looked at it and said, "This is really cool! Why don't you show it to Microsoft?" They all said the same thing. And I go, "Well, yeah, I don't want to show it to Microsoft, because they're busy doing their own thing. But this would be a really good opportunity for you as a publisher to take to market." And they go, "Ah, you know, I don't want to fight with Microsoft. I don't want to get in that pissing contest." So *finally*, you know, so many people stood there and said, "I really like this, why don't you show it to Microsoft," I went back to my buddy who had taken me to the Bank of America. And I had met and gotten to know Bill Gates a little bit in the early days. But we weren't friends, and we didn't correspond. So I went to my buddy, Glen, and I said—this is in 1988—I said, "Can you get me an audience with Bill Gates?" And he said, "Let me work on it." And he called me back a couple

weeks later, and he said, "Okay, you're not going to see Gates, but you're going to see one of his guys. Come on up."

So I came up, and the guy who I had the interview with was a guy named Gabe Newell... I went into my spiel, giving my little demo, and it's a half-hour demo, and about 5 min into it, he just pushed his chair back, and he goes, "Bill's got to see this." He didn't want to see any more of it. He knew he was looking at something cool! I said, "Great!" I went home and had a meeting with Gates a month later.

So you bet I coded like a crazed coding weasel for a month, adding cool new features to the prototype, and I show up at Microsoft, you know, go into the HQ Building, and I'm ushered into a big board room with a big table, and about a dozen Microsofties pour in, and I've got my little computer in front of me, and I start demoing this thing to Bill. And it blew his mind! He'd never seen anything like it. At one point he goes, "How did you do that?!" I go, "It's magic, Bill!" I mean, what do you say when he asks something like that? And it was when I showed him sprite animation. 'Cause nobody at Microsoft had done that yet. I mean, you have to BitBlt onto the screen, and it's complicated, and I had to figure it out and build all the routines to do that. And then I was showing him something else, and he looked at his guys, he goes, "Why don't we do stuff like this?!" I mean, it was really interesting. I had no idea how wrong that was, but I learned later. And at one point, one of the guys starts criticizing it, and says, "Well, it doesn't do this, and it doesn't do that."

And Bill, as I was about to leap to the defense of my program, *Bill* leapt to the defense of my program! So at that point, I knew, this was gonna happen. Eventually, signed the deal to deliver this shell construction set. It was gonna be the front face of Windows 3.0, which became the first successful release of Windows. And I wrote the product to completion, and delivered it through their—through Microsoft Quality Assurance Process. That was again back in the days of Golden Master disks, not of continuous delivery. And the—it got caught in a political battle within Microsoft. Microsoft was fighting with IBM, who was their big patron at the time. And Windows was actually not a strategic product for Microsoft. OS/2 was the strategic product, and it was for IBM, their client, and it was getting all the calories within the organization, and it was the B-team was working on Windows. But the B-team kept looking at the 640k barrier. And they said—they broke it! And they broke it before the OS/2 guys did, because the OS/2 guys didn't consider it to be an issue. But the Windows guys knew it was. And so Windows really stole a march on OS/2 and, thus, there was this huge pissing contest within Microsoft.

None of this that I was party to, because I had brought significant technology in from the outside, which embarrassed a lot of guys. A lot of those guys who Bill sat there in that meeting and said, "Why can't we do stuff like this?" I didn't realize at the time that what he was doing was he was making all those guys at the table hate me. You know? Because, you know, I showed them up really badly. And so the shell construction set, they said, "Look, you have to be able to be identical to the OS/2 shell." And I said, "Well, look, you can build the OS/2 shell from scratch in about ten minutes using Ruby." They said, "Is it keystroke for keystroke identical, and pixel for pixel identical?" And I said, "Well, it's close!" Well, okay, that was just enough of a beachhead that they could point to it and say, "This won't work." And so they kicked it out of the build. And it did not go out as the shell construction set for Windows 3.0. And it became an orphan within Microsoft. And it bounced around looking for a home within the organization. I flew back up to Seattle and met with Bill and I said, "Will you sell it back to me? 'Cause I'll release it myself. I'll publish it myself as a shell construction set for Windows." And he thought about it and he said, "No." I had no leverage, and he figured he could do something with it. So I came back home and I tried to start a company, and of course, I was seriously nondisclosured.

And so I struggled for a while to try to figure out what to do. And finally within Microsoft, Ruby, this visual programming front-end, which I had always seen as a tool for users, found this sort of strange bedfellow of QBasic, Microsoft's BASIC, their interpreted BASIC, which at that point was a dead product. The hobbyists in the world were using Pascal and the pros were using C. I wrote Ruby in C. And I was not a fan of BASIC. I've never

been a fan of BASIC. Anyway, they put BASIC together with my visual front-end, and released it as Visual Basic, and they invited me to the rollout, and I went up to Washington and watched, and I sat there seething with anger in the front row.

I mean, it took me a while to overcome my—"What have they done to my baby?!" kind of attitude. But it was a huge hit. It was a huge success from the very moment they released it. At the time, the microcomputer was ascendant and the mainframe was just at the apogee of its life. And there was this enormous cohort of mainframe programmers who were looking around going, "Oh, shit! We're in trouble!" And to program in any of the conventional languages, they would have had to have just given up everything they know and made this arduous journey to the microcomputer. And along comes Visual Basic, and it was just this easy hop from COBOL to VB. And so it was this enormously empowering tool for this cohort of mainframe programmers to make the change over to writing software for Windows. And the other thing that I did in Visual Basic was it had this palette of controls. And you could drag-and-drop this control onto a Window, and then imbue them with behavior. So we came up with this bright idea of having those controls be dynamically loadable. And so each little control was written as a separate code module that communicated kind of at arm's length with the main program. And so what it meant is that you could write, completely separate, a control, and build it as a DLL, as long as it knew how to respond to certain messages and behave in certain prescribed ways. So what would happen is Ruby would come in, and it would connect to all of its internal controls, but then it would broadcast to anybody saying, "Anybody out there know how to speak Ruby?" And if a dynamic link library recognized it, it would say, "Yeah, I'm one of your guys!" What it would so is it would say, "Oh, well, tell me your name, tell me your icon, tell me your functions." And it would suck that stuff in and it would extend the little palette, and then you could click and drag on it. So it was this third-party aftermarket thing. And this was just revolutionary. Again, this is—it wasn't that the technology didn't exist, 'cause it was there.

Microsoft built it. It's just that nobody knew how to use it. Nobody thought in terms of—they thought in terms of, "I'm building my functions for me." And I was kind of thinking, "What could we do to create places for others?" And this was new and novel. So, when this interface was released to the world it was called the VBX Interface. The Visual Basic Extensions. But what it did was create a commercial marketplace for third parties. This is what they could do is they could write these DLLs, and they would just work. And it was one of the reasons why VB became so popular. Because I had populated it with some very rudimentary tools, and all of a sudden people came along and started writing really sophisticated tools that communicated with the protocol. And it would be part of VB. So you could click, and drag-and-drop a very competent spreadsheet in the middle of your window. That kind of thing.

So the fact that it was really the first integrated development environment. You know, where you did development inside of a program that could help you, and it was visual. That was, again, my design, it was a visual programming environment. I mean, Microsoft took that much, farther. They developed all kinds of really cool things and ways to step through the language. It was really neat. But the basic idea that you're programming inside a visual environment, that was mine. And the basic idea that it's dynamically extensible and open to third parties, that was also my idea. And I think those were the two really significant contributions that made Visual Basic a success. And so a guy that I had known from way back in the 1970s, who had a—who was an author, a technical writer in Marin County in the mid-1970s, named Mitchell Waite, was—had been writing books about software, he was interested in this field. And he had actually started his own publishing company, called Waite Publishing, which he eventually sold to McGraw-Hill. He through, I don't know what connection it was, he got called up to Microsoft and they disclosed Visual Basic to him early on, and they said, "Would you like to write a book?" And he said, "You bet I would!" He was really intrigued by Visual Basic, this idea of this powerful programming environment.

So Mitchell Waite wrote the very first book in support of Visual Basic, called, "The Visual Basic How-To." And I think it came out, not with VB, but a few months later. But I hadn't talked

to him in a decade, and I got a call from him one day and he said, "In the "About" box of Visual Basic it says Cooper Software. Is that you?!" And I said, "Well, yes. Yes it is!" And he goes, "Can I buy you dinner? I want to hear this story." And so I said, "Sure," and we had dinner in San Francisco, and I told him the story. And he sat back at the end of the story and he goes, "That makes you the Father of Visual Basic!" And, "Well, I guess it does!" And so he dedicated the book to me, "The Father of Visual Basic." So he gave me my one-phrase resume. I became the Father of Visual Basic, and that opened a lot of doors at that point. A lot of people that I had been saying, "Oh, I did something big over there," they didn't believe me. And then people started coming to me, and saying, "Is that you?" <laughs> So it was sort of gratifying to see that. And my opinion about VB, you know, began to change. You'd kind of go, "All right," it's actually the metaphor "Father of Visual Basic," is actually really an appropriate metaphor. Because as the Father—if you're a father of children, you know that you make a contribution, you do your best, you support them, and then they go out and they surprise you. Your kids will surprise you. And so Ruby surprised me by becoming Visual Basic. But it became a huge success, so I'm okay with that.

Visual Basic, I sold that to Microsoft in 1988. And that was pretty much the end of it. <laughter>... It was later in that year, and then it was finally released as VB in 1990, I think? I shepherded it to delivering at Microsoft, and then they took it to final release. This is one of those things about this—I don't know if it's the tech world, or if it's the world in general, but the things that are a success are kind of accidental, and weird combinations of people contribute to them, and you never know where it comes from. And the stuff that has turned out to be big and prominent in my life are things that are not necessarily the stuff that I would say is the most significant that I've done. The things in my life that have given me money, have not necessarily been the things that I thought were worth money; and the things that I thought were worth a lot of money, some of them have just quietly slipped beneath the waves, and not made me a nickel. And it's just—it's a funny world out there. The things that people come around and say, "Oh, Mr. Cooper, you're the greatest! You've done this amazing thing!" I look at it and go, "Yeah, I knocked that off in a weekend!" And the things that my heart and soul are in, nobody knows about. It's one of the conundrums of this world, certainly of my world. So you learn as you get older, you learn a kind of equanimity, just move on. Whatever! That's the thing about innovation.

Hsu: ...Was visual programming something you came up with on your own, independently of any other influences?

Cooper: Well, yeah, it's not like visual programming didn't exist. It was—that I kind of pushed the boundaries. And yeah. That's what I did. I pushed the boundaries. And making it extensible was a *big deal*. I mean, again, like I say, dynamic link libraries was something Microsoft invented, and it was based on computer science that had been around for many years. So again, nothing really earth-shattering about that, except if you go to 1988, and you look at the DLLs, there were none! If people wrote something as a DLL, they did it as an exercise, then statically linked it, or in effect statically linked it. I was one of the few guys who actually used it.

Hsu: All right. Let's move on to interaction design. So it seems like a logical step to go from making Visual Basic and this shell development environment to then actually moving into this higher level of looking at the actual principles of design.

Cooper: After I shipped Ruby to Microsoft, I went off and invented another product, and learned a new word: monopsony.

Cooper: We all know the word monopoly. It means when only one company is selling. Monopsony is when only one company is buying, and as a guy who is trying to invent stuff and sell it to publisher, when I did SuperProject, there were probably 200 software publishers out there of equal stature, and I offered my product to them, okay? When after VB in 1990, when I took Flute [Cooper's subsequent project] out into the marketplace, there was Microsoft and Adobe and a couple others. And so they could just say, "No," and I would just twist in the wind, and I remember I took Flute up to Microsoft and Bill goes, "Yeah, we're working on that." They still haven't. But so I'll tell you, the first product

that I shipped, I mean, I wrote a sort program and a name and address manager <inaudible>, but the first real product that I built at Structured Systems Group was a General Ledger, and it was pretty bad.

I mean, it was awesome because it was, you know, the first viable accounting program on a microcomputer and it sold just fine and I built a business on it. But, you know, looking back on it, it was all I knew. I came from a batch processing COBOL mainframe background, so I built a batch processing BASIC mainframe program to run accounts, and, you know, the paradigm has moved on, well beyond that. But I knew it was hard to use and I got some complaints from people. I learned so much building that General Ledger that I then went on and did an Accounts Receivable, Accounts Payable, Payroll and an Inventory system—so we ended up with a whole suite of accounting stuff. Well, the second package was the Accounts Receivable, and I really took pains, I was really proud of it, because it was better. I took all the problems with the interface in the General Ledger and I fixed them all in the Accounts Receivable, and it was a significant achievement and it was much awaited by our clientele. Everybody who owned a GL wanted to buy the AR, and we started shipping it, and people were buying it and they were very happy with it, except they started complaining, and they said, "All the commands are different." And I said, "All the commands are better," and they said, "But they're all different," and that's when the light came on.

As I realized I could hear somebody knocking on my frame of reference. All of a sudden I knew there was something new going on here that I didn't know about, that I hadn't conceived of, that my self-referential design was missing something. There was something big there, and I didn't know what it was, and so this was in the 1970s, and that's when I changed my focus and I started to think not about from the software out, as you would build it, but from the user in as you would use it, and so that began my path. Now I was still an inventor and a developer, but everything I did after that was done from the point of view of how would users use this, and you have to understand in 1978 that was a radical notion and other people weren't doing it. It took me a long time to begin to get to where I had some command of this stuff, and Visual Basic, Ruby, the visual programming language, was a user-centered thing, because it was based on this idea. I mean, I had identified users in the field who would use it and configure it for who and what and its behavior. I mean, it was still very self-referential and very programmer-oriented, but for its day, it was pretty out there in terms of its ability to be easy to use. It's one of the reasons I think why Visual Basic was such a commercial success. But it was finally, after I couldn't sell Flute, I ended up—

Hsu: Flute was the. . .

Cooper: Flute was the product that I worked on afterward. It was really this kind of information management system. It would manage your e-mails and your names and your appointments and stuff like that, calendaring systems. That I began to realize that I was, that what was really interesting to me was this idea of thinking about what software should be and how it should behave, and that was much more interesting to me than actually writing it, which I had been doing for the last dozen years and had kind of. . . I was kind of done with that, and I had a real crisis in my life about, I thought, "Well, nobody would hire me to just design products because anybody who designs products builds them, and anybody who builds them designs them and you could never just design the product," and I really kind of moped around the house and whined for a long time. <laughs> Until I talked to some friends and they just encouraged me to try it.

So one day I was at a conference somewhere, I was leading a panel, and on the panel were several of my friends and colleagues from the development world, and at the end of the panel, the audience filed out and I looked at my panelists. I said, "Hey, you guys. You're the first to know. I'm a consultant. I'm not going to do any coding but I'm here to help you make your products easier to use and better," and a couple of them hired me. To my shock and amazement, they did. They hired me because they knew me as a programmer, they had faith in me, and so I began to help them without programming. So, I would look at their product and give them advice and make suggestions and redesign interfaces, and that's how I went from being an independent software

author to being a consultant, and really it was interaction design. I didn't call it that. I called it software design at first, because I still thought like a programmer. But stopping programming was really a remarkable thing. I'd been programming nonstop for so many years through my entire career, and to stop all of a sudden really gave me a lot of insight into how the way programmers think affects the way software is designed, and, you know, over the next couple of years I built up a pretty good clientele. I had several people I was working for and they liked my work and I started to get really busy. I started raising my rates and I still was busy, and finally in 1992, I went to my wife and I said—Sue, you know, has had her own career in tech marketing and she was the director of marketing at Logitech back in the day and I said, "Honey, come on my rounds with me. I want you to see what's happening," and she came around and she found that I had, like, a fan club of people who liked what I did, and we got our heads together, we realized that we, there was a business there if we wanted it, and it took some soul-searching, because we realized that this was a big step forward and it would be hard to step back. But we, we decided to create a company and she started to help me on the marketing and sales side and we ended up hiring a designer, a brilliant young guy named Wayne Greenwood, and he showed up for work Monday morning. I was still working out of my home office and Wayne <laughs> kind of showed up and would come wading through the kids and Sue and I looked at each other and said, "Oh, yeah. We need to get an office, don't we?" <laughs> You know, because we weren't thinking of it the way startups are thought of now as it's like a hothouse and you're incubating somebody who hopefully will get acquired by Google.

Back then we were thinking in terms of actually doing good things in the world and... But the company grew. We were very lucky in the early days. The name Interaction Design barely existed. There were no jobs anywhere, in any company, called User Interface Design, Interaction Design, Design, nothing. It was just a green field, and so to hire people, we had to just kind of thrash through the underbrush and people slowly... I wrote a book. I mean, I learned so much starting that company. I learned so much. I'm not a writer; that's not my thing. But I learned so much in such a short period of time doing this stuff that it was, from the time we really began, it was three years, and I wrote a big, fat book called *About Face*, of everything I knew. I put it all in there, every last thing. When I was done, I said, "That's it. I don't know anything else," and I said, "I'll never write another book, because I just put it all there," and three years later I wrote another book, because I was learning so much at the time. But that first book really, it really struck a chord out there in the world and a lot of people were trying to figure out how to solve these same problems, which is they had technology and people weren't happy with it. It was good, strong, powerful technology. People were unhappy. This is a familiar feeling to a lot of technologists and it was rampant in the 1990s, and they found this book, *About Face*. It's one of the few books that actually talked about it and actually had some solutions, and it was very practical. You know, there was a lot of—there was academic stuff where it talked about analyzing usability after a product was built and how to test a product that's already built, but nobody talked about making a product easy to use before it was built, okay, and so they would call me up and say, "Could I come work for you?" and so we built this cadre of really sharp men and women and who had very checkered backgrounds and oddball resumes and many of them are out there in the industry today doing significant stuff in big companies and some of them are still with us too, and so we were really the first.

Hsu: Could you talk a little bit about the principles that you espoused in your books and how you arrived at them, and the impact that that's made?

Cooper: Well, what's interesting is that design has all these different roots [like a tree]. It's more like a bucket of rocks as there's the usability thinking comes in here and then there's the web design thing comes in here and then there's the typography guys come in here and then there's the industrial design thinkers come in here, and there's all these different constituencies with their own different motivations and I was the really, the only guy, who came from a

world of software development. Who entered this world with that history that was, it was just different, and. . .

It needs to show you who it is, what it's doing, and how far along it's going, okay, and it needs to be nice to you. Which means it needs to anticipate your needs. Like, I came up with just this fundamental notion, which is, goes like this, and it's still remarkable how often it's forgotten. I said, "If it's worth the user entering, it's worth the program remembering." The thing about software is every time it's loaded into memory it's a *tabula ra*sa, and so software would come in and it would, no, it—"I don't know anything." <laughs> You know, and I would say, "No." You know, if the last time you ran the user moved the window up here, the next time it runs the window needs to be there. Stuff like that. I mean, just none of this stuff is rocket science and it's all fairly widely accepted today. At the core of my thinking, early on, it became very clear to me that the way software is built is, it's built functionally. So this is how all software is conceived of, and it's how all software used to be designed and it's written and still an enormous amount of software is still all about the function. Here's the functions we're going to add, and. . . But that isn't how users think. It's not how people think, and so they—people have a different way of conceiving of things. They think about it in terms of, "Where do I want to be and why do I want to get there?" and so I began to see that there was this fundamental dichotomy. There was thinking about tasks and there was thinking about goals.

Too many people approach the problem of interaction design as interface design. So they said, "Okay, how can I make it easier for you to pump gas in your car on your way to visit Grandma?" and I looked at it and said, "Your goal is to visit Grandma, not to pump gas." So is there a way to make it so you don't have to pump gas at all? Those are the kind of questions you need to ask. You see, interface design works at your gas tank. Interaction design works on getting you to Grandma. So I called my methodology *goal-directed design*, and it was based on this notion that if you know who your user is, what their desired end state is, and why they want to get there, then you can create well-behaved software that will make them very happy, and if you come at it and you say, "Okay. As an organization creating client management software, I want people to keep the database pure," you start thinking in terms of editing fields to keep the database pure, and it has nothing to do with the needs of the salesperson out there who's trying to manage his or her client contacts. You see, it's so easy to start designing and building software from the software out, and it's so hard to design it from the user in, because you have to put yourself in the shoes of the user, and so much of the design of the behavior of software was vouchsafed with engineers and this was a problem. Because engineers are the ones building this thing and what you're doing is you're saying, "Okay, Mr. Engineer, you need to get into the heads of your user," and they go, "yeah, I got that." But they wouldn't, and because to get into the heads of the user, that's not an engineering job, and it doesn't use engineering tools or an engineering mindset. It's a compliment to engineering, okay, and so this led me to where it was very clear that there was a problem in the industry is that organizations were structured in such a way that they were hiring a bunch of programmers and saying, "Will you guys design it?" and design it so it's user-friendly?

But the problem is, is that that isn't what programmers do, you know? Asking a programmer to be user-friendly, to put themselves in the shoes of the user, is really playing against type. It's not something that the developers are really particularly good at and, and so I ended up writing a book called, "The Inmates are Running the Asylum," talking about this. Saying, that, see, what I had done was I had been a developer and I stopped developing and it immediately became apparent to me how much the way developers think influences the way developers think about interaction design. . .But the thing about designing from the user's point of view that they don't get is in order to do that you actually have to go out and listen to the users, and it turns out that programmers hate listening to users, and so whenever developers get jealous about that role, they go, "Oh, I don't need a designer to tell me what to do. I can figure it out myself," I say, "Oh, are you going to go out and interview users?" No. They're not going to go out and interview users. They're going to tell designers what they

do from their ivory tower. Well, I've been in that ivory tower for many years and I know how ridiculous that is and how it doesn't work and as soon as I put that trap in front of them and they fall into it then they realize, "Oh, yeah. That's right. I guess somebody does have to listen to the users and it isn't going to be—so I guess I'm not the interaction designer around here."

The problem is that management tends to want to hurry everything up because they're still thinking industrial. You know, is that if we're more efficient in our development we'll be better, and they don't understand that efficiency doesn't have a role here anymore, and effectiveness is much more important, and so building software faster means nothing. Building software better, that means everything; and by better it doesn't mean that the functions are executing smoothly, but that the user is arriving at their goal, and I used to play this game with people in groups where I'd say, "Well, what's your goal here?" and they'd say, "Well, my goal is to, as a user, is to get this data entered," and I said, "Okay. Well, how about if you take your clothes off and enter the data?"

They go, "I'm not going to do that." I go, "Right. That's really not your goal, is it? Your goal is to keep your pants on." There's—I probably can't say that anymore like I used to. It's probably viewed as sexist and horrible. But my point is that there's a whole bunch of tacit goals that precede the corporate goal that we don't acknowledge, okay, because we tend to come at it with our rationalist programmer's point of view, and so we said, "Okay, the job here is to fill this out or to record my contacts or stuff like that," but that isn't the goal. The goal is to not be made to feel small, to not—to be made to feel bad, to not sit in front of a computer and go, "What is it saying to me? I don't get it! I push the button and it's not working!" That's the number one goal. People don't want that. So the fact that you can contact their clients is secondary to the fact that they don't want to be made to feel like an idiot, so if your software contacts all their clients but makes them feel like an idiot, then you don't have a success and you don't have a good design. This is revolutionary stuff in the mid-1990s. Now it's kind of seen as, "Yeah, that's about right," although people still are a little shaky about how to get there. They still tend to say, "Well, if it's got cute stuff on the screen, that will do it," and no, it won't. But that's, you know, how I arrived at the basic principles of design.

Now, then comes the question, "Well, how do you get there?" and there's a whole bunch of other stuff that derived from that, because this is what we were doing. Clients were coming to us and they were giving us complicated domain problems and we had to solve them and we realized early on several things. Number one is, you can't imagine what people want. You have to go out into the field and you have to find real users and you have to listen to them, and you can't ask them—you don't do it with questionnaires. You know, if you go out in the field and you find somebody who's using Oracle every day and you say, "Well, how do you like this Oracle software?" They say "It's great. I love it." You go, "Well, what do you like about it?" "I like this thing here and that thing there." And then you say, "Hey, want to get a beer?" You drink a beer and you say, "So you really like your job?" "Oh, yeah, yeah, it's nice and. . ." but, you know, and after a while they're going, "You know, I hate this Oracle software. It's terrible. It's just—I hate it. The other day, I was just—I just broke down and I was sobbing at my desk." This is real. We've had stuff like this happen, or you'll have somebody will be sitting there and they'll—you say, "Are you able to memorize all these commands?" "Oh, yeah, no problem." Then, an hour later, you turn their keyboard upside down and it's covered with Post-its and cheat codes.

What I'm saying in this kind of thing, for interaction design, there's not a lot of value in quantitative stuff, but there's a lot of value in qualitative, and qualitative, you have to just ask nothing but essay questions and then you have to shut up. You ask me a question. I rage on for a while and then I stop and then instead of jumping in with a question, you wait. And then you wait one more beat, and then you wait one more beat and I jump in again, you see? That's what a good listener of a good qualitative interview does and that's what you have to do in the field. Know-it-all programmers—and God knows, I'm a know-it-all programmer. I can't do that, okay, and so it's a really hard lesson to learn as an interaction designer to go in the field and be

patient and listen and listen and listen some more and ask open-ended questions and get people talking, but after a while, you earn their trust and they start to spill their guts to you, and you can't—if they give you suggestions, you have to discard those because they are users and while a broken clock can be right twice a day, they're still not dependable.

I was working with a client one day and I was just having the hardest time communicating to them the results of what I was working on, and so just as a kind of an expediency one day, I said, "Okay, I'm going to say there's this person named Carolyn." I think it was Carolyn was her name, and she wasn't a real person, but I described her job and it was totally representative, and I described how she uses the product, which was totally representative and I said, "And this is what she wants to accomplish," which was totally representative, and when I presented that to the client, they said, "Yeah, we know that person. We've seen a hundred like her." And then I said, "This is what she needs." And that's how I invented "personas," because it's a—while it's a hypothetical archetype, it's based on the qualitative research that you do in the field and it's really a bucketful of characteristics that you've observed empirically, and what you can do is use this person to define who the user is, what they're trying to accomplish and why they're trying to accomplish it, and it's immensely powerful because it allows you to gather your thoughts together and to encapsulate them in such a way so that you as a designer can manipulate them in your design, because you express a persona as somebody who exists because of their goals, and their goals exist because of the persona. It's kind of a tautology, and then what you do is—now, you have a purpose, right? There's what does the product do? What would the user want to do with it? Well, what you can do is, you can then begin to write a series of scenarios. The person wants to use the product to accomplish this end for this reason, and so then what you do is, you as a designer, you say, "Okay, well, here's the screen and here's the buttonology." Then, what you do is, you take the persona and you play act, role play the persona using your proposed solution and say, "Does it get them to their goals without making them take off their pants?" And if it does—well, if it doesn't, then you go around again and it's nice and cheap and free. You can do it on a whiteboard or on a sketchpad. It's easy, no code, fast, and you can do it a hundred times if you want, and when you find, yeah, that does get them to their desired end state without embarrassing them, then you know you've got a good design; so, as a design tool, it's really second to none. But the thing that I discovered that is so powerful about design personas is that a funny dynamic takes place in the boardroom or the meeting room, which is, as a designer, you don't code. And if you don't code, you're half a person.

Okay, in the Silicon Valley culture, coders have two votes and people who don't code have one vote. Okay. So if you sit in a room and say, "As a designer, I have studied. I have researched. I know this is what I think we should do," all the programmers are sitting there going, "Yeah, until you have facts and we just have to go with opinions, we'll go with mine." That's how they're thinking and—but a remarkable thing happens when you say, "I've done the research, I've been in the field and here's Carolyn. She's our representative user, and Carolyn wants you to do it this way." All of a sudden, the programmers go, "Oh, well, we can do it that way," because it's not you and your opinion. It's a psychological tool of incredible power because it's really hard to go up against a programmer, because programmers are really smart and they're lawyers. I mean, they argue cases in front of the central processing unit, who's more nitpicky than any judge, so if you're going to—personas are a great tool for just kind of finessing that argument and so they're very powerful tools. We used them widely across the company. I wrote about a bunch of other tools that we used, but this is the one that just galvanized the industry and a bunch of people jumped on the bandwagon and started writing their own books and their own blog posts and stuff about it, and—but very few of them got it right, and so actual—in terms of pure word count, a lot of other people have written more about personas than I ever have and—but a lot of them talk about personas as mechanisms for describing program functionality, whereas I invented them specifically for getting

away from program functionality and they're just misdefined and misused [it]. We've done projects for clients without personas, but we tend to find that they're incredibly useful. We use them all the time, so that's one of our very successful tools that came out of trying to invent a practice.

And another one of the tools that we use is one that still has not widely caught on. It's pair design. We've done pair design since the early—well, since—yeah, since the early 1990s, we've been—we—early on, we learned that there's something magical about two people working together when you're doing this kind of creative work, where you're trying to solve problems by innovating, and we find that—I mean, people working by themselves can certainly have good ideas, but they can also start breathing their own exhaust and start losing perspective, and when you get large groups together—like, there was a craze for brainstorming there for a while. But the problem with large group brainstorming is, the loud people take the day and the quiet people don't, and it's really—you get people shooting down ideas and one-upping and political stuff, and in our experience, large group brainstorming is not worth the bother or the expense. It is very expensive to get a lot of people together. And one person working is good, but two people—there's something magical about two people, there really is, when they're mutually supportive, because it's really easy to take criticism one on one, whereas one on two taking that same criticism gets really personal. If you look at somebody and say, "Yeah, I think there's a better way to do it," and there's somebody watching, it's just—that hurts, but when there's nobody watching, the other person can go, "Yeah, how would you do it? Let's work on this," and—so now, we have taken that much farther, so there's a whole bunch of nuance as to how we do that pair designing, but we've done it pretty exclusively for 25 years now and we think it's one of the most powerful design tools, and I've had people—it's so funny. They say to me, "Alan, what's your secret?" and I say, "Pair design," and they go, "Yeah, well, we can't afford that." But what can you afford, bad design? What business are you in? Are you in the business of saving money or are you in the business of making lots of it by creating products that people really like? It's not more expensive. I mean, the cost of having two people do the design of one is not high compared to the value you get when you create something that people really like. So we—early on, we found that when you get two really creative—it's like two alpha dogs. When you get two alpha dogs doing pair design, they tend to spend a lot of time arguing about "my way." Ask me how I know that. <chuckles> Wayne and I used to argue all the time and we'd find ourselves—we'd have these long arguments and then at the end, we'd finally realize that we were in violent agreement. And so we began to realize that there was something more to this, even though we were still so very productive with working as a pair, that we were not the optimum match, and so our first thought was, "Well, we need a junior designer," so we hired a junior designer and we discovered that we had a designer [that] wasn't very good and they weren't much help at all, and so then we said, "Okay, really what we're looking for is somebody who's like a technical documentation person, who can be the amanuensis for the designer," and so we hired a technical doc person, and they literally ran screaming from the room because it's just—it's—technical documenters are in the—historically what they do is, they take what is and document it, and what we were looking for people to do is to take what isn't and instantiate it. And then, we realized that what we were looking for was a designer, a skilled, intelligent, capable, experienced designer, but somebody who looks at things differently.

And we finally got to this—we now called them generators and synthesizers, and they're kind of arbitrary terms. The idea of a generator, that's the alpha dog. That's the person who's always running to the whiteboard and saying, "It should look like this," and the synthesizer is the person who sits back and watches this for a while and then says, "How do you solve this case?" And the generator goes, "Oh, shit, okay." They erase it. "Okay, we're going to do it like this," <chuckles> because the generator can have 10 ideas in a minute and a synthesizer is the one who starts riffing through all the scenarios and say, "How do we do that? How do we know that this is going to work, how this is

right?" and it turns out if you put a gen and a synth in a room with a bunch of other people, their abilities, it just depends on how assertive they are, but you put a good gen and a good synth in a room together, just the two of them, they really can move forward and they can do an immense amount, and the gen is constantly saying, "Where's the brilliant idea here?" and the synth is constantly saying, "How do we integrate all this stuff so that it becomes a coherent whole and work?" and the gen is saying, "What's going to make this thing sizzle?" and the synth is saying, "How are we going to communicate this to the world?" and the two really work together. Now, we have hired people who are pure synths and pure gens, but we also have people who are—who can go back and forth and people who—you know, and there's—we've got a lot of really sparky, inventive synths, and lots of really thoughtful, reflective, critical gens, so we're not—when we began doing this, we saw this as a job and we now see it as a role. And so when you work at Cooper, you're a Designer or a Principal Designer or Senior Designer, and you don't—you're not—whether you're a gen or a synth is not in your job description. You know, we'll talk about it in your role on a given project or engagement. So these are the kind of tools that we've developed for doing this kind of design work and there are some companies out there that do pair design and are very, very successful with it, and there are a lot of companies out there where we talked to them and their designers come to us and they go, "I'm so lonely."

Hsu: Okay, I have two more questions. My second to last question is how did you get involved with Agile software development methodology?

Cooper: Well, I'm not really sure that I am involved with Agile because my days as a developer are long gone, and Agile is a development methodology. There are some people who think that Agile is a design methodology, to which I say, "No, it's not, and if you think it is, you deserve all the hell that's going to come to you, not from me, but from your users." Agile, on the other hand, is a remarkable and unique thing and I'm a huge fan of it. It's the—you know, I used to joke that every seven years, the community of software developers rises up as a group and throws a tantrum and smashes all their toys and decide that, no, we don't want these assemblers. We want compilers, and then they say, "No, we don't want that. We want object-oriented," and they say, "No, we don't want that. We want…" oh, structured was one. "First we want structured, then we want object-oriented, then we want reusable," so the thing is, all of those tantrums are all about techniques and tools. And then along comes Agile and it starts talking about the developer's responsibility to create a product that is successful, and it's the first tantrum that says, "Let's talk about people and process," so it's a qualitative difference amongst the community of developers for the first time in 40 years, or 50 years, and so it's remarkable because of that. I don't think it's a coincidence that one of the underlying principles—one of the tactics, I should say, of Agile is pair programming, the notion that we're not trying to be efficient; we're trying to be effective, and so they put two minds on it. I was a big opponent, in the early days, of Extreme programming because it was brought to me as a design methodology and I just kind of said, "That's cray-cray. That can't be," and I kind of dismissed it, and I didn't pay attention to it, and it was Lane Halle—that's her maiden name. She wasn't our first synthesizer, but she was the one who really wrapped her head around the role of synthesizer and really developed it. She's a very intelligent, capable woman. She used to work at Microsoft before she worked at Cooper and now, she worked for a while for Pivotal and a couple other companies, but she's in New York now making copper cookware and I think she's doing some other tech stuff on the side. And she grabbed me (this was 10 years ago) and she said, "Alan, Agile is not Extreme programming. I want to take you out and introduce you to some people," and she did. She took me out into the world and introduced me to some Agile practitioners and really changed the way I thought about it. I realized that it wasn't just another name for Extreme programming. Extreme programming—there was a lot of—I mean, the whole Agile/Extreme came out of a lot of frustration the programmers had, because programmers had been told, "Go this way and you will be successful." They went that way and they

got good salary, but they created stuff that they weren't proud of and they were unhappy with that, and they felt blamed, and Extreme programming was a way to say, "Well, if we're going to hatch a catastrophe, you're going to hatch it with me," but Agile said, "What can we do to not hatch a catastrophe?" It was much more positive and so it was something that I've been a big fan of. I call what we do in the interaction design world "responsible craftsmanship," because you have to take responsibility for the end result and it's a craft, and it's the exact same phrase that I use to describe Agile, real Agile, not rigid three week scrums and stuff like that, but real Agile is responsible craftsmanship. It's developers taking responsibility for creating successful products and making people happy, not just creating a bug-free program.

Hsu: At the beginning of this interview, you mentioned that you actively made certain decisions to not become Bill Gates, and those decisions that you made encapsulate sort of your philosophy or your—the way that you see, you know, being a good software developer, being a good designer. Could you talk about some of those decisions and how those decisions showed what your values are.

Cooper: I'd be Bill Gates if I could. When I first started out, like I say, I wanted $50,000 a year. I thought that was everything, but then as I began to learn more, I realized that there was a lot of money involved. I could build a big business. I could make a lot of money, but the decisions that you're faced with every day, it's a lot of kind of little decisions and I found that all the little decisions I was making were—each individual decision was small and easy to make. It's "No, I'd rather take this path than take that path," but the sum total of those decisions sends you down a different path than the path that Bill Gates went down, and you. . .

Hsu: Are there any examples that you can give?

Cooper: It's like when we worked with the Windows guys. I just so was so not proud of that code, and that code has made billions of dollars. And the code that we delivered, I'm proud of. I'm not saying it was the greatest code ever because it's not. It certainly had its shortcomings, but it was as good as we could make it and that stuff that we saw coming from Microsoft wasn't like that. Well, when you make code that's really good, that's something you do for yourself. It's not something you do for money and that's kind of the difference, so let me say that when I made conscious decisions not to be like Bill Gates, what I mean by that is, I've made unconscious decisions to not be like Bill Gates, and it's—I realize—I've come to realize—and you could certainly accuse me of post facto rationalization, okay, and I would not argue with you, but I say that money people get rich in money and idea people get rich in ideas. You know, money people have a lot of money because they value money in everything they do and they pay attention to money in everything they do and they think about it first and foremost and they're not afraid to look you in the eye and say, "How much money do you have? I want 10 percent more than that," whereas an idea guy, who—I'm an idea guy. I go, "This is a really good idea. Let me rub up against it and get all warm and fuzzy with this wonderful idea and okay, oh, look another idea. Let me go rub up against that idea," and so I've got to do a lot of really interesting work in my day. I've worked with the big guys. I've done business with Bill Gates and I got to code sitting next to Gary Kildall. I've written books about what I've done. I'm winning a Fellow award from the Computer History Museum. It's—this is what I have to show for my work. This is the result of the decisions that I have made. And I certainly—I could've taken any one of those products that I invented over the years and taken them to market, but from the very beginning—at SSG, you know, I conceived it, invented it, wrote it, code it, documented it, shipped it, supported it, market it, sold it, yammered on the telephone to dealers across the country. I did it all and so ever since then, I've been trying to do less, have a smaller wedge of the pie, and there's a cost to that. The cost to that is, I don't make the big bucks, but what I do is, I get lots and lots of different slices of really interesting pies. You know, Silicon Valley has been really good to me and I'm not complaining and I'm doing just fine. I've gotten to do a lot of stuff. Mostly I've gotten to be really self-indulgent. I get to play with the toys that I like playing with, and sometimes those toys make a lot of money and sometimes they

don't. They never really made a huge amount of money for me, but it's not like I want. I do just fine. I live in paradise out here in the country and I have a lot of friends and I got kids and grandkids. So it's good.

Hsu: Thank you. Is there anything else you would like to add?

Cooper: Yeah. There's one thing I'd like to add, and that's that we've reached a new level of responsibility as technologists —I call it our Oppenheimer moment. Robert Oppenheimer was the inventor of the atomic bomb. He ran the Manhattan Project, and he had his head down making the finest weapon he could and when he saw that bomb explode, all of a sudden, he kind of went, "Oh, shit. Is this really going to give me what I want in the world?" And that's where we are today. This is Silicon Valley's Oppenheimer moment. We've created these machines that extract data from us and give us advertisements in return, and the mechanisms of civil oppression used to be guns. Today the mechanisms of civil oppression are Facebook and Google and Uber and we're creating those tools, us technologists and designers, and so we have to all of a sudden go, "Well, wait a minute. Is this really what we want to build? Do we really want to create this nuclear blast?" And so there are engineers out there who are saying, "Oh, I'm in a. . ."—young men and women's saying," I'm going to go to Silicon Valley and I'm going to be one of those 500 startups and I'm going to be the next Google or the next Facebook." It's like, think this through, because there are people who'll write you a big fat check with a lot of zeroes. All you have to do is screw over the human race. So, is that who you want to be? Do you want those zeroes that badly? This is really the answer to your earlier question <laughs>. And as I look at the problems of our society today and I try to deconstruct them and say, "How did we get here? What's the problem?" I find that each one has the same root cause, and that's inequality, is that I find out here I'm learning about the food chain and I'm discovering that the food chain is distorted. Well, what distorts it? Well, what distorts it is inequality. I look at the information economy of Facebook selling my soul to Russian election hackers and the root of that is inequality, is that there are people out there who can afford to buy my story. And so when you're sitting there in Silicon Valley and you're saying, "I want to make a billion dollars," what you're doing is, you're saying, "I want to create more inequality." Okay, so I've been an entrepreneur all my life and I've always wanted to create my own business. And I wanted to get rich when it was $50,000 and then I wanted to get rich when I thought I could make a few million dollars, but when I was offered the opportunity of joining Microsoft as a key contributor in 1988, I told them no. Okay, I'd probably have a lot of money today if I had done that. Those are the kind of choices that you make where you say, "No, you know, I don't want to work for that company." I don't want to do things the way they do things because I don't like that. I don't think that what you do is you amass enough money so then you can be the world's greatest philanthropist. What I'd rather do is have everybody join together and do something collectively. I think it's better for the—I trust our collective judgment better than I trust Bill and Melinda's judgment, and so responsible craftsmanship today—when I thought up that term, what I meant is being nice to your users. Make them happy. It will be good for everybody and you will have a successful business. Now I say it has a greater meaning than that. It means—responsible craftsmanship means that every artifact you build is another brick in the wall of the concentration camp or a brick in the wall of the cathedral, or a brick in the wall of the great tower honoring ourselves. It could be what you want it to be. But if you say, "Oh, the brick is agnostic," believe me, people will build concentration camp walls out of it. They will take advantage of your "I don't care," so that's my parting thought. Thank you, Hansen.

Hsu: Thank you very much.

Hansen Hsu is a historian and sociologist of technology, and curator of the CHM Software History Center. He works at the intersection of the histories of personal computing, graphical user interfaces, object-oriented programming, and software engineering. Hsu received his PhD in Science and Technology Studies from Cornell University in 2015.