Oral History of Alan Cooper

Interviewed by:
Hansen Hsu

Recorded March 13, 2017
Petaluma, CA

CHM Reference number: X8126.2017

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. We're honoring Alan Cooper as one of our latest fellows. So the first question: What were your most important life lessons?

Cooper: Hi, Hansen. Thank you. I get taught life lessons all the time. Probably the most important one is that people will tell you [that] you can't do something and you have to ignore them because you can, and most people have more power than they think and I remember being told the things I couldn't do and it was important to learn to ignore those. I suppose another life lesson: When I wrote my first book my agent said to me words I'll never forget; he said, "Alan, it's your first book. It's not your best book" and I found that that was very useful in so many things, is you have to forgive yourself as you're learning.

Hsu: Great.

<crew talk>

Hsu: What was your proudest moment?

Cooper: Well, my—I would say my proudest moment—moments were the births of my two sons and my long-term partnership with my wife. I mean I'm—that's not a moment. I mean I could say I was proud of getting married but really I'm prouder that we've been together for 37 years and we've built something together. We've created a community, we've created several communities, and that's really what matters.

Hsu: Thank you. What were the turning points for you in making your life decisions? What inflection points happened in your life that led you down the path that you took?

Cooper: Well, my life is filled with inflection points. I think most people tend to go on a trajectory and I—my life has been more episodic. I dropped out of high school and everybody, my parents and my teachers and my counselors, they all told me that I was going to be a terrible person living a miserable life and when I quit high school I said, "Okay. I'm going to go live that life of being a ditch digger" and I set out to fulfill their prophecy, and it took me a few years of growing up and being independent that I realized that they were wrong. And I came back and I said, "I'm going to reenter society and I'm going to enter it on my terms" and I set out to be—to go to school to be an architect, and I had this crazy idea of paying my way through architecture school by programming computers. And as soon as I got my hands on a real computer—a real digital computer all of a sudden I fell head over heels in love with the digital world and that was a huge turning point. All of a sudden it's—I mean architecture remained a goal for me that I pursued for a while sort of fruitlessly but software was where I did real architecture. And I got a job and did the normal stuff and then I started my own company kind of on a—it was almost a dare. It was like wouldn't this be cool if we could do this and when personal computers, microcomputers first came out I discovered that I could buy a computer with money I had and put it on a table in my house and this was just a revolutionary turning point. And I was not an entrepreneur looking for a business to build; I was a creative guy who discovered that I could have ultimate creative control so that was another huge turning point. I would say that another significant turning point was when I stopped programming and realized that I really wanted to focus on the behavior of software and how people reacted to it. And programming is mind absorbing and I love that about it; there's nothing I like more than wrapping my head around a complex, difficult, demanding project and just getting lost in the complexity and the mastery of it. And so programming was a natural for me and when I stopped doing it it was this—the—somebody threw open the windows. It was a wonderful, refreshing thing and I saw how much it dominated my thinking, but what it did is it released me to think about this—the issue of user interaction design in a new and fresh way. And so I have to say that letting go of things is a turning point; it's an enlightening step. And so there have been a whole series of those in my life. I leave them behind and move on to the next.

Hsu: Great. Thank you. What might have you done if you had not followed one of those turning points?

Cooper: Well, as a kid I wanted to be an architect. I didn't know who I was or what I was and one day—I don't know how this happened but one day I was about 13 years old and I just had this bolt of inspiration and I knew that I wanted to be an architect; that's what I wanted to do and as a kid that meant really designing houses. And then I started putting all my energy and attention into architecture and house design and as my—as I got a little older and my interests matured I realized the thing that really fascinated me was urban planning and really transportation planning. I was very interested in the notion of how our built environment tends to follow our transportation mechanisms and so I got really interested in fixed guideway transit systems, and I realized that they were—they are the lifeblood of our cities and they're far more effective and more efficient than an automobile-based system and that was something that really consumed me. I still am very interested in that and I still think that our destiny is some form of railroads. As a guy with a challenged academic resume, <laughs> I wanted to study; my goal was to study architecture and urban planning at the College of Environmental Design at UC Berkeley and—but with a 1.8 GPA and no high-school diploma it was a little problematic to get in, and I tried to do it. It's possible to do it if you get essentially a perfect score on your SATs and I got good SATs but not perfect and so they said good-bye to me, and I went off and kind of hitchhiked around the world for a while and lived an alternative lifestyle until I finally came back and I reapplied to go to the College of Environmental Design. And I had at this point taken a lot of community-college coursework so I had kind of showed that I could do the work and CED to their credit not only accepted me but they gave me a full scholarship. Unfortunately, it was the exact same time that I started my first company and so I had this horrible decision of, do I continue with what looked like a really promising business, an entrepreneurial business which was fraught—it was filled with unknowns—or begin this long course of study. And so I went to the university and said, "Will you put my acceptance on hold?" and they said, "No" <laughs> but they said, "You can apply again" and so I did; I applied again and I went back to work. And in this ensuing—I guess it was semester—the—all of a sudden I—we brought our first products to market, I started making money, and they accepted me again; only this time my full-boat scholarship, which was needs-based, was reduced to a hundred-dollar honorarium <laughs> but they accepted me again and—which was a great thing. And I was torn but at this point I knew I had to continue with this company and so I sent them back their hundred dollars and thanked them and I actually applied a third time, figuring that this—the—I had no idea. I was a young guy and entrepreneurism wasn't a big thing in Silicon Valley in 1976 or '77 and they accepted me a third time and I have never attended a day of class there. And it's delicious irony 'cause I've been working with those guys lately and they love that story and I'm looking forward to possibly teaching there at Berkeley next year so—but really the—architecture's what I wanted to do, and of course the irony of the whole thing is that my contemporaries who went into building architecture do less architecture than most software designers. There's not a lot of architecture being done; I mean you're either Frank Gehry or you're some minion in a giant firm doing takeoffs for walls and rooms, that kind of thing. And in terms of urban planning I mean that is just—you are just at the bottom of a political pyramid and you have no authority and no power and nobody really wants to do long-term investments that it takes to do things like build fixed guideway transportation systems. So I'm fortunate that I found a place where I could do architecture, I could create a built environment in a digital universe, but I think that's where I would be otherwise.

Hsu: Wow. Thank you.

<crew talk>

Hsu: So looking back what do you see as the impact of your contributions?

Cooper: Well, the impact of my contributions are—it's—I wrestle with this because what we all want to say when we look back is we say, "I did this." "I freed the slaves." "I did some significant signal thing." "I invented Ethernet." I don't have something like that to look at. Instead I just—what I see is that I was a pioneer in areas that masses of people have subsequently proven were the right path. So I committed utterly to the microcomputer and I committed to it in an entrepreneurial way, so in 1976 I remember going around and telling my friends and family, "We're all going to be owning these computers" and people would go "What? I don't want a computer. What would I do with a computer? That's stupid." And we all have computers, hundreds of them in our lives now, and so I just realized that this was going to be so at a time when there weren't very many people who had that certainty.

<crew talk>

Cooper: So it was a commitment early on to this notion that computers that people had was going to be a huge giant defining thing, was significant and then I worked really hard on a whole bunch of stuff in that definition and it's turned out to be true. So what's my impact? I don't know. I was one of a thousand soldiers but I was at the front running as hard as I could. I did the same thing in the world of—I mean it became clear to me as a builder of software for people to use, that building it was one problem but making it so that it was nice for people to use was a wholly other problem, and I committed to that problem early on and worked as hard as I could. And it's now clearly evident that that's a big deal and so what is my impact? Well, I was one of those loud voices early on who said that it was going to be a big deal. Did I make it a big deal? No. Am I the only guy who said it was a big deal? No. Was I just loud and obnoxious about it? Yeah, that's me. So, now within that there's a bunch of things that I have done that I look at with pride and accomplishment that I was able to build companies and give people places where they could learn and expand their own abilities. I mean it's a uniquely—it's one of the benefits of being an entrepreneur, of waking up in the middle of the night and <laughs> crying about something that's not working right. That's the downside but the upside is that you get to look at young people and say, "Without me they would have struggled to do what they're doing" and so that's a rewarding thing and especially in the—excuse me—especially in the early days of the interaction design world. I mean there's always—back in the microcomputer days there was plenty of jobs for programmers, always have been, but in the early days of starting to think and work in terms of interaction design there were no jobs; there was no notion of it as a profession either and so—and it wasn't a title, nobody—and so there was just these strange people who were kind of orbiting around the technical world. They were either programmers who didn't want to program or they were testers who didn't like testing or they were technical documentation people who didn't like documenting or they were any one of the above who knew that that—those were not the solutions to the specific problem but where could they go to do what they knew they had to do, which was focus on the design of the behavior of technical systems as they are presented to humans, to users. And I created a safe place for those people and it's—this is a—I think a significant contribution and a lot of those people have gone on to roles in big companies, in Apple and Facebook and Instagram and other companies like that, and they're leading the charge there and they're writing books and they're running conferences and I kind of go "I had a little bit to do with that."

Hsu: Great. Thank you. Who were your role models?

Cooper: My role models are—and that's a really interesting question 'cause I don't—it's been a long time I think since I really apprenticed myself to somebody. In the early days, I guess in a lot of—in some ways my first role model was a guy I went to high school with, a guy named Gary Fisher who's a very creative guy, and we were really on the outside. I was on the outside just 'cause I was a kind of an obnoxious, annoying kid and he was on the outside because he was a—it was interesting—he was a bicycle-racing prodigy who had been kicked out of the sport for being countercultural. And so for four years in—the last couple years of high school and a couple years after we did psychedelic light shows for rock and roll bands in San Francisco and Marin County and it was just an—a wonderfully, delightfully creative moment and it was entrepreneurial. We tried to make a business of it and we made a pittance and we were treated like dirt but it was creative with a capital 'C.' It was so much fun and we invented out of nothing an art form that was—used projectors and film and liquids and images and it was just amazing and it was so much fun; it was crazy. We made stuff up, we did wild things, and it was just that brief moment and it all went away and—because there was no—there was really no way to make money at it and so our light show ended, it just kind of petered out, and I went on and threw myself headfirst into the digital computer world. Gary went on to invent the mountain bike; he's a famous guy and he's a real kind of ambassador to the world of bicycling and—but it was just fortuitous that we were a couple of high-school buddies. When I started to study programming I was really lucky to get a teacher at the College of Marin, a guy named Dan Joaquin who was a—he wanted to be a practitioner and teaching was kind of a side interest for him I think, but he was a teacher unlike any teacher I ever had because all the teachers that I'd ever had were teachers out of academia and because I had never been to a university they weren't even really the cream of the educational crop. And Dan who had been formally educated was really—came at it from the point of view of a practitioner teaching me very practical stuff, how to, and it was really eye opening and he shaped the way I think about programming. He was the guy who introduced me to the arts and mysteries of programming and I'll never forget the—one of the classes that he taught was IBM basic assembly language and he came into the class on opening day and he held up the little IBM quick reference card, it was a little four-by-eight-inch card, and he said, "This entire class is— everything we're going to learn is contained on this little card" and then he—a dramatic pause and he let it unfold. And that in fact is what it was and it was one of those gestures that made me think that this is going to be good and it was, and so he was very much a mentor. When I got into the microcomputer field Gary Kildall was certainly a mentor. Gary was the brilliant professor who delighted in seeing the interest awaken in his students and he was a brilliant man and the industry owes an enormous debt of gratitude to Gary. And it was—in many ways it was Bill Gates who cashed in on the work that Gary did but Gary was the guy who invented so much of what we know of microcomputer software technology today, and I got to work with him for a few years and that was—it was inspiring. Gary also inspired me in—he's famous for being an airplane—a small-airplane enthusiast and we had a meeting once in probably 1978 and he flew up from Monterey where he lived and landed at Oakland and—to have a meeting. And he said, "Well, let's go meet in the air" <laughs> so we went flying and I was sitting in his little airplane and then I went "I'm going to do this," and a couple years later I started taking flying lessons and became a private pilot and flew small planes for 25 years and I have about 1500 hours and I got my instrument rating and I've flown quite a bit, not as much as Gary, but he was very much a role model for me as a hobby aviator. In the late '70s at Structured Systems Group, it was just me and another guy; we were software geeks and we needed help with marketing. And we hired this young guy who's a couple years older than me but he was our first director of marketing and his name was Dave Carlick, and Dave was just a funny, witty brilliant guy who was—he was a copywriter and an entrepreneur and he worked for us as the head of marketing for—really for less than a year and then he went off to start his own advertising agency that became the very first digital ad agency in Silicon Valley and—which he grew and eventually sold to some big ad agency and— but Dave was a real mentor to me and showed me a lot of, a completely different side of the way to behave in the tech world. As his parting gift to me when he left Structured Systems Group is he introduced me to his kid sister in 1979 and I married her and that's my wife, Sue, and we've been business and life partners for a long time since then. So Dave became my brother-in-law and so I tell these brother-in-law jokes like comedians and—but it's really true; Dave is a—has been an important mentor and role model in my life. I would also say that Bill Gates is a role model. I thought I was going to be Bill Gates for a while 'cause our companies started about the same time and they grew at a similar pace in the early days and I began to realize the decisions that I would have to make to be Bill Gates and I found myself constantly not making those decisions. And years later when I did business with Bill he asked me to join Microsoft and I turned him down because it was not the way I wanted to live and not the way I wanted to work, but he was still a prominent person in my thinking.

Hsu: Thank you. What advice would you have for current and future generations?

Cooper: Well, the advice I would give has changed a lot recently. I would have said in the past, "Don't listen to the naysayers. Do what you want to do. Create value." If you set out to make money, you—I'm not interested in that. I'm not interested in people who want to make money but if you want to make something great that's really interesting and I think that if you make something great you'll make money. You may not become a billionaire but you'll do just fine and that's what's of interest to me; that's something that I respect in the world. But recently it's become pretty clear that the digital tools that we've been creating are more socially influential and more socially powerful than we ever expected. And so now what has become clear is that we have a huge moral responsibility that was not evident until just not too many years ago and so the advice I would give today is "Are you working to become a good ancestor or are you just working to make money?" Because you can make money by burning villages down and rebuilding them; that's a good way to make money but it's not a good way to be a good ancestor. And so you have to ask different questions. It's not about is the behavior of my software user friendly and is it bug free and robust and powerful but now you have to be saying, "Is this the right thing to put into the world?"

Hsu: Wow. Thank you. Why do you think computing history is important?

Cooper: I think computing history is important because I was in it <laughs> and so everybody wants to be recognized but the—I love to read history and who was it who said it's—history doesn't repeat itself but it echoes? And so I think it's important to know the echoes because it does and what happened before is going to happen again, and technologies are horses that we ride and when you're astride a galloping horse you begin to realize how powerful you are and you begin to look at all the people who are not thundering on your war charger and you begin to look at them as somehow inferior and it changes the way you think and it changes your value systems. And you study history so that you know what that sense—that sensation of power really means and it hopefully keeps you from being seduced by that power and being seduced by that idea that because you are a captain of industry or a captain of technology that you are in fact not some kind of special person. And I think that there's a broad myth adrift, a meme, in Silicon Valley that it's a meritocracy and that it's—that the people who succeed are the people who've worked the hardest and who are the smartest and all that, and one of the reasons we study history is to make it as abundantly clear as possible that it is in fact not a meritocracy and that you do not rise to the top because you work hard and because you're smart. I know dozens and dozens of hardworking, smart people who are not rising to the top and what—you rise to the top because you're lucky and you rise to the top because you have a support system of people and society, civil structures that give you the leg up, the advantages that you need. And there are a lot of incredibly deserving people who don't have those advantages and they don't rise up and history shows that the people who are successful in Silicon Valley are the people who start out with a trust fund and wealthy parents. And my parents weren't wealthy but I grew up in Marin County, one of the wealthiest suburbs in the country, in the world, and my parents were the poorest people on the block but they were on the block. I was the beneficiary of that mid-century largesse of America and I was able to survive 'cause when they told me I was going to be a dropout ditch digger I realized that that was a choice I had and it wasn't a predestination. So that's why you study history because you see people's heads get filled with their own story and then they bring ruin on the world and we study history to get a little humility so that we realize that we have a responsibility. This is certainly something I've learned here in 2017 is that—growing up in mid-century Marin County is that I thought that I could—I didn’t have to participate in politics; I didn’t have—my civic duty was nil because the structures of civilization were doing just fine without me. And what I’ve learned recently is that if you don’t participate, if you don’t contribute, don’t pay attention and you don’t give back then it can go away in the blink of an eye, and so I’m rededicated to this notion of no, if you want a democracy it doesn’t mean you are—it doesn’t mean you live in a democracy. It means you participate in it or you lose it. That’s been a hard lesson, and, again, history is where you learn that, because then you can see it in its perspective.

Hsu: Yeah. Thank you. Why do you think the Computer History Museum, specifically, is important?

Cooper: Thank you to the Computer History Museum. I am profoundly and deeply honored to be a fellow precisely because I honor the Computer History Museum because they don’t lionize successful business people. They lionize successful people, people who have done significant things. I like to think I’ve done significant things, even though I’m not a billionaire. And I’m not saying that the billionaire shouldn’t be honored, but certainly not—I don’t think you should be honored because you’re a billionaire, and I don’t think that you should be honored—I don’t think the honor should be solely bestowed on billionaires. And I think the Computer History Museum is more egalitarian than others, and they’re more cognizant of the value of the technical contributions, and I feel the same way about academic work. I think that academic work is really interesting, but I’m much more interested in the people who are—maybe it’s my entrepreneurial approach to things. It’s nice to know things and to think about them and write about them, but I’m more interested in what you do about them. And it’s really easy for an organization to be seduced by people who make a lot of money, so I think that’s what separates the Computer History Museum, and that’s why I’m really proud to be here.

Hsu: Thank you. This is great. So now we’re gonna sort of take a step back and go back to the beginning of the story and start [to] tell it more chronologically. So, of course, the very first question there is where and when were you born?

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 did you get into the counterculture because that was sort of what your friends were into in high school?

Cooper: You know, I had another buddy in high school and we've sort of drifted apart. We were inseparable for a few years together there, and I used to—I thought Roger was the bravest guy in the world, and he was kind of intimidating to me, because he would do really kind of crazy things, and I'd follow him. And he pushed me. We met at a friend's reunion, you know, twenty-five years later, and there was Roger, and I hadn't seen him in years and years and years. And we started telling stories, and he announced to the crowd there. He said, "Alan was crazy, and he did all these amazing things, and he pushed me, and he challenged me." And all of a sudden I had this revelation, this is what teenagers do is they don't know, and they follow each other. They follow each other in circles. And so I thought I was following Roger and it turns out Roger was following me. I'm sure we were both, but that's what you do. That was scary when I heard that. I went, "Oh, No."

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 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 twelve year old kid I'd sit in the back of the class with a bunch of twenty 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: Wow, thanks. 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 pre-architecture at a community college so that I could get into Berkeley, and pre-architecture 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 desi—I call it "Alexandrian design" after Christopher Alexander, who happens to be a professor of architecture, or formerly professor of architecture at Cal Berkeley. And he talks 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 fourteen 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. It was all very technical, technique, and 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. And what was the question again?

Hsu: Oh.

Cooper: Where am I going?

Hsu: We were just talking about hobbies.

Cooper: Oh, hobbies. Well, I love hobbies. I mean, to me in many ways programming and design are hobbies, you know. They're just, 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. That's a very interesting aspect to it that most people are unaware of. For a long time I was really, really involved in paintball, and, in fact, I was involved in it before it was even called "paintball," and nowadays paintball is a high speed, open air sport. And back in the day it was really much slower and more strategic. We didn't have repeating paint markers. They were single shot markers, so it was about stealth, and planning, and interesting teamwork. But I spent many, many, many happy weekends in the Santa Cruz Mountains dressed in camouflage, crawling around on my belly trying to shoot my best friends. Ha ha. I got into high powered model rocketry for a while, and went out to the Black Rock Desert. We'd have huge encampments out there, very kind of—this is long before Burning Man was started, and we'd get a waiver from the FAA, and shoot these model rockets. There's a couple of them hanging up there [in the ceiling of his barn/workshop where we are recording]. That guy's been launched at Black Rock, and there's some others. To a model railroader who makes little static, little thing, you know, you landscape with tweezers, model rocketry is really interesting because it's all about transients of energy, you know. It goes from standing still on a launch pad to all of a sudden being subjected to thirty or forty positive Gs, then zero Gs, then thirty or forty negative Gs, and, you know, loss of air pressure, and humidity and temperature, and so it's an exercise in dealing with shocking transients. And, of course, the inclination is to build protection in. As soon as you build protection in, you build weight in. Soon as you build weight or mass, then you need more complex systems to protect against them, but as soon as you have more complex systems, you need more mass, and you need more complexity, and so you begin to under—you get an insight into why, you know, NASA is the way they are, and so it was very educational. As I said earlier, Gary Kildall introduced me to flying, and I spent many, many hours in the cockpit of a small Cessna, or Piper, Beechcraft plane flying. In the early 2000s I finally had a little bit of money. I bought a nice airplane, and I started flying. I had my instrument rating, and I did a lot of flying. I flew across the country several times, and it was a real adventure, you know, flying in the northeast air space. I flew into Boston Logan in my little plane. That was an adventure, and I landed at Teterboro about three weeks after 9/11. I could still see smoke rising off the Ground Zero. It was an interesting experience. And I've had a lot of fun. I decided it was time to stop flying. I just kind of had—flying is one of those hobbies that takes—if you have ambition as a pilot, it takes more money than common sense would allow, and my wife finally slapped me upside the head and said, "Is this really what you want to do?" And so after a forty year career building software, having my artifacts being the intangible world of bits, I found myself really craving making things out of atoms. And I had learned from my dad, you know, some rudiments of woodworking and metalworking, and so I started making things out of wood, and actually because I liked woodworking, but I knew the real challenge for me would be metalworking, I ended up taking a year of night school as a machinist, and then another six months as a welder, to learn things that I didn't know, to challenge myself. And so now that we live on the ranch, I took this old barn and converted it into a workshop, and it's immensely roomy, and spacious, and generous. I call it "The Shop Mahal," and I've got a full machine shop over there, and a metal fab shop, and a sheet metal shop. And I've got—it's a wonderful place to work. And then in this side is a woodshop. I've got a full milling capacity, and fabrication, and all the tools, and dust collection, and shop air, and it's a real pleasure to work in it. So that's what I get to do. I get to spend a lot of time making things with my hands. Some people really like to create things. I like to create things with my hands, and I think it's why I've never built a big company, because at a certain point you have to stop making things, and you have to start working through others, and I'm much more interested in the things than I am in the others.

Hsu: Earlier you mentioned one of your hobbies was model rocketry. 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 '60s was antithetical to the way I learned. I had a saying when I was into model rocketry, and I would—you know, we would do a fundraiser for the kids' school, and I would donate a model rocketry class that the parents could spend a little money on, donate to the school, and I would take a half dozen kids and we'd have a one or a two-day workshop making model rockets. We'd spend the first day doing some experiments, and then we'd make little model rockets, then the next day we'd go out to the launch field and we'd shoot our Estes rockets. It was a lot of fun for me and for them, and I came up with this axiom, 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, is to me mathematics is the burden if you want to make cool things. And, you know, beyond multiplication, I just kind of—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. I love that notion.

Hsu: What things were you into reading as a kid?

Cooper: Well, as a kid, I read a lot of science fiction as a kid, and what we think of now as the classics, Asimov and Heinlein, and Frank Herbert, Dune was big. But I kind of stopped reading science fiction as I got older and read more history. I'm kind of omnivorous. I read—I like genre fiction, and I love nonfiction, guys like John McPhee, and just interesting stuff about interesting things, Simon Winchester, and I love Alan Furst. I love his—I mean, he is absolutely a genre writer, but he's so good at his genre. I like guys like Patrick O'Brian, and… Alan Furst, Patrick O'Brian. I've read a lot of mediocre authors and a lot of good ones. I tend—I probably read two-thirds nonfiction, one-third fiction, because it's harder to find good fiction, I think.

Hsu: What kind of nonfiction do you like to read?

Cooper: I read a lot of military history, and historical as well. Also recently since moving to the ranch out here, I've been reading a lot about the new agriculture, and the failure of the old agriculture, and that's been kind of an eye opening thing to me.

Hsu: So earlier you mentioned that you dropped out of high school. Could you elaborate on that story, and was it related to your countercultural activities?

Cooper: No.

Hsu: No.

Cooper: No, not at all. No. It was related to the—no. The high school was a highly rated, proud high school in an affluent suburb, and they had a vision of themselves. It was how they were. And I did not fit that vision, and they worked hard to get me to fit there, you know, their square hole, and I was a round peg. And okay, so here's the defining thing about me. Tell me what to do and I will do anything except that. That's kind of me in a nutshell is I do not like to be told what to do, so if you tell me you can't do that, I do that, and sometimes to my detriment, and sometimes to my benefit. But in high school I did things differently, and in high school your job is to do things the same. And they felt that it was their job to discipline me, so the year I—I went to high school for four years and dropped out in the middle of my senior year, really. One of the things that I did in my senior year was run a seminar, a workshop, on alternative education with students and a couple of teachers who were interested. So I was active and working hard, but I was, you know, and they knew I was smart, and all that, but I would not toe the line. And the year after I left high school, they started an alternative high school that a lot of the younger kids coming along went to, and they did a lot of the things that I advocated that I got in trouble for advocating. And I'm sure that there are a lot of people back then who would confuse cause and effect, but, no, no. I was—I mean, I wanted to be a good kid. I was a polite, good kid, you know. I held the door for people, and my parents brung me up right, but, yeah, when people say this is what you got to do, I would do the opposite. I'm still that way, you know. Tell me to keep my voice down and I'll start yelling. I just—it's just—it's in my blood. That's the way I am, and sometimes that leads me to great discoveries and big accomplishments, and sometimes it leads me to be unhappy. That's life.

Hsu: So then after you left school, you mentioned, you know, you could have been a ditch digger. How did you just make your way in life? How did you decide at that point, you know, what direction you wanted to go?

Cooper: I got a job washing dishes at a restaurant, and I did light shows on the side, and I had a lot of fun. I mean, Marin County in 1969, that was the summer of love. It was pretty exciting. I was kind of weekend hippie. I'd go into San Francisco. There'd be, you know, happenings in the park, or I would hitchhike over. I had a bunch of buddies who were in my cohort who did go to UC Berkeley, and so I would hang out at UC Berkeley, and sometimes drop acid with the kids over there, and get teargassed on, you know, Dwight Way. And in 1972 a buddy of mine said, a guy I went to high school with, said, "I'm going to Europe for the summer. You want to come with me?" I was terrified. I had never been anywhere, and I didn't speak any language, except English, but it just sounded too good to be true. I had to go with him, and he said, "Don't worry. You'll come with me. I'll show you the ropes. I know French." He didn't know French. And I thought, okay, this is going to be good. And so I bought my ticket, and I had a motorcycle. I sold my motorcycle, and I put that money into traveler's checks. I was ready. I had a backpack. I had long hair and weird clothes, and stuff. And a week before we left he said, "Look, a friend of mine has a small airplane and he's going to be flying across country. I'm going to go with him. I got family in New York. I'll stay there for a week, and then I'll meet up with you in London," because I had a ticket to London. I was just so terrified, because all of a sudden now I'm traveling by myself to a strange foreign country, you know. I knew nothing, but my ego was at stake. I couldn't back out because it would have been a huge loss of face, and I really did want to go, but it was a great experience for me, because I had to face my fears. I had to get on that airplane. I had to fly, and I had to land in this—with jet lag. That was back when they gave you vaccines, and so I had, like, diphtheria, the twenty-four hour diphtheria, some horrid disease. I don't know what. Anyway, here I am sitting kind of almost in tears on a street corner somewhere in London somewhere, and all of a sudden because my whole vision of the way things were going to be, because my friend had said, "Oh, yeah, I got a friend. They'll take care of us. Here's their address." He gave me their address. And I knocked on their door. There was—nobody answered. I looked in. The house was vacant. So I was just bereft. And then all of a sudden I kind of went, wait a minute. That's just your image of who you should be, and what you should be doing, and how this trip should be, and it's not going to be like that. And I stood up and I said, okay. I can do this. It's not that hard. I said, "Okay. What I need first is some sleep." And so I found a place to spend the night. I woke up and by the time my buddy finally arrived a week later, I said, "I don't want to travel with you. I want to be by myself. I don't want you helping me speak French." And so we made—we decided to go completely different ways. I started thumbing north, and we made an appointment. We said we'd meet on the fifteenth of July, or something, in Tivoli Park in Copenhagen, and we did, and it was great. We spent the night drinking beer and telling stories, and stuff, and I ended up hitchhiking all around, taking trains around, having a great time, meeting people. I got a job eventually working as a carpenter in London. My buddy bought a bicycle and rode bicycle to Rome, and we had all these adventures. And when I came home, I was a different person. I had learned so much about myself, and about who I was, and what I could be, and what I had to be, and I came home. The first thing I did was I enrolled in college, and I took the hardest courses I could find. I said I'm going to figure this out. And they told me that I could take a—I just walked into the college and I said, "How do I go here?" They said, "Well, let's see your high school transcript." I said, "I'm a dropout." They said, "You can take a GED." I said, "Why didn't they tell me that before?" Nobody had ever told me about the general equivalency in all my years in high school. I was shocked. I said, "Okay, I'll take it." I signed up for the test, and I took it, and when I was done, they said, "You passed. Now, you can petition for a high school diploma." And I said, "Over my dead body. I will never have a high school diploma." And I'm ridiculously proud of that. Because it's—that was my path. And one of the things that—it was the weirdest thing is I was in Zurich, Switzerland, and this was in 1972, when there were—when computers were big mainframes, and they were in glass houses. And I was walking down the street and there was some Swiss bank, and there were these low windows looking down into the basement, and there was a giant IBM 360 computer, and I stood there spellbound, looking at that computer with all those switches and all those lights, and I said, "I want to learn what all those switches and lights are for." And it was like I was in a trance. I walked in the front door of the bank. I was still a longhaired hippie. This was a Swiss bank. <laughs> And they looked at me like, "What are you doing in here?" And I, kind of, went, "Oh, I guess, I better go," and I ran out. But when I got home, that was what I did is I said, "I'm going to learn--" and I finally did learn what all those switches and lights, that had so impressed me, were for. They're there to impress people. That's their job. They serve no other purpose. <laughs> And modern computers don't have them. Those original old computers had those panels full of lights and that.

Hsu: So this was at the College of Marin that you enrolled?

Cooper: Yeah.

Hsu: And were you just taking general classes? Or how did you eventually decide you wanted to do architecture?

Cooper: Well, I had wanted to do architecture since high school. But I couldn't get into a university, and so the College of Marin was, sort of, a side door, and I had this idea, well, I thought I could get a part-time job programming to pay my way through architecture school. Of course, there’s really no such thing as a part-time job programming, and as soon as I started programming, I said, "This is it. This is what I love."

Cooper: 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 at the College of Marin. The College of Marin's computer, they had a 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.

Hsu: How did you get your first real programming job?

Cooper: Well, I had a buddy who said—he was working as a programmer in industry, and he just told me I was over qualified and I should go out and get a job, and so I did. I put together a resume that was all over inflated, and I put it out there, and I got a job working as a COBOL programmer in San Francisco. I worked for American President Lines, a shipping company, and when they gave me that job, I walked out of the interview going, "Woo boy, did I fool those guys." And it wasn't until a few years later that it dawned on me that I wasn't fooling anybody. They knew who I was. They could see, you know, through my second hand ties and my long hair that I hid from them. They knew that I was a crazy guy, but that I was going to work hard for them. Because I knew I was a crazy guy and I knew I was pulling one over on them, and therefore, they knew I was going to bust my ass, and so it was one of those things where I just got hella lucky, because they put me in charge of a giant program. At the time, they had all these batch processing programs that ran this company that owned, like, 30 ocean freighters that went all over the world, and this thing was called the Cargo Handling Standard Cost System. And the communication was primitive, and if a ship, you know, would pull up in Singapore and load and unload goods, they didn't get the actual stevedoring bill for six months. So what the company had to do, in order to maintain its books, is they would get a scratch telegram that would tell, you know, "We unloaded 40 tons and we loaded 30 tons." And what they'd do is they'd say, "Well, based on past performance, loading and unloading this many times was probably about this much and therefore, we'll book it that way, and then next year, we'll do an adjustment." Okay. So that's what this program did. So it had to have all these different currencies and da-da-da. It was a big complicated thing. And so I spent my time reading old COBOL source code written by bored programmers who—I mean, this is old spaghetti code and stuff. And one day, I discovered this—I looked at it and I said, "Wait a minute. That's not right." And I started playing it through in my mind, and I realized that what they had done was they—the way the code was supposed to work, it was supposed to read a record, then loop a hundred times, processing it, then read the next record, then loop a hundred times. And what they had done was they had read the record, looped, read the record redundantly, and looped, and there was this redundant clause inside the loop, and I looked at it, I said, "This is a redundant read." And, you know, the reading of a record off a disk is incredibly slow, and I go, "This can't be right." And I tested it every way I could, and finally—I mean, you had to, like, apply for time on the mainframe, and so I went to my boss and I said, "Look. I want to run this in parallel." And he said, "Okay." So I just commented out one line of code. Okay. They had a huge 370 mainframe that cost millions of dollars and had people operating it, and I commented out one line of code, and the Cargo Handling Standard Cost System, instead of running in four hours a day, ran in forty minutes. I was a hero. <laughs> And so even though I was the long-haired hippie that everybody goes, "Look at that crazy guy," all of a sudden, I'd made my bones at American President Lines, and so they really liked me. We got along fine, until the company got taken over and I ended up—I quit to start a software company. And they looked at me and they said, "You'll never get a job as good as this one in data processing." And I said, "Yep, you're right." And that was my last job in data processing. But...

Hsu: So what was your first exposure to microcomputers?

Cooper: Well, it was...

Hsu: What year was that?

Cooper: Boy, you know, there was a kind of a culture of—there was a timesharing culture. Now, our computer, being a batch process business data processing IBM 370, it was not a timesharing system. It was just a single batch process. But there was, kind of, the mini-computer timeshare thing was growing, and I spent some time at the Lawrence Hall of Science in Berkeley, where they had timeshare 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 8008 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 mini-computer accounting system? And, you know, instead of spending $30,000 on a PDP-8 or something with an accounting system, is that 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. Instead, my buddy and I each, you know, emptied out our bank accounts and started a company. We each put up $7000 and that's how we started the company. And we did exactly one turnkey system to this one accountant, and a few months into it, we, kind of, went, "Why are we selling the hardware? Everybody can buy the hardware over-the-counter. Why not just sell the software and let them put the software on the computer?" And so that's what we did. So this accountant who bought this first system, we sold it to him in, I think, it was '77, and I ran into him 14 years later. And I said, "Do you still have that computer?" I said, "If you've got it in a basement or something, I'd just love to have it, just for fun." And he said, "Well, we're going to take our last client off of it in a month or two." He was still running it 14 years later, which shocked me, immensely. But he said, "But you can have it. It's just going to be scrap iron for me." So he gave it to me, and I have it. It's over there in the other room, and it, actually, still works. I mean, I threw the floppy disks out and all the complexity and stuff, but you can still toggle a program in from the buttons on the front panel.

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? Your involvement in the counterculture, how important was that in getting you exposed to the early microcomputer hobbyist scene?

Cooper: Well, it wasn't—<laughs> 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 straight out of est. One of the first developers whose software I published at my first company was a guy who came out of straight-laced Indiana, and he was a very flamboyantly gay guy, and he came to San Francisco where he spread his wings. He was a good programmer, an inventive guy, created a great program, demanded that it all be—all of its trade dress was purple, and he was just a different guy. You know, everything about him was unconventional, and actually, it was interesting. There were a lot of guys from Indiana, in the early days. There was a whole cohort who came out here, and some of the real original thinkers and writers in the era. There were just a lot of different guys. I remember, later, in 1982, I worked for a very short time for Gary Kildall. I took a job in Monterey, at Digital Research. It didn't really work out that well, and I ended up leaving. I've never been able to hold a job for very long. But in that time, I was the applications guy in a world of systems guys, and so anything that had to do with applications, I was involved with. And one of the things, they were exploring a partnership with an applications company that had been making a big splash with their spreadsheet and their word processor. And so they called me into a meeting with some guy that was the head of this company, and he was standing there in his suit and tie, and giving this chalk talk about his products, and I was just looking at this guy going, "I know this guy from somewhere." And I couldn't figure out—I'm listening to him talk, and finally, it struck me and it was so incongruous that I stood up in the middle of the meeting, and I pointed to him and I said, "I know you. You're Buck Truck and we used to drop acid in Berkeley in the '60s." And it was a really inappropriate thing to say <laughs> in the middle of a business meeting. And he said, "Let's talk about that later, Alan." And he went right back to his spiel without missing a beat. But that was what it was like in those early decades. There were a lot of people who had a long history of not doing what they were supposed to do, said, "Hey, look at this new computer stuff. This could be a greenfield for me." And they came into it. I mean, you know, Bill Gates dropped out of Harvard, not really a great height at which to be a counterculture guy. But then, Steve Jobs dropped acid and went to India and lived an alternative vegetarian lifestyle, and there were a lot of guys like that, who—it was the, you know, wearing bell bottoms and having your hair long was, kind of, the equivalent of having computers and playing with them and doing wacky stuff. It was like being a creative musician. "Look at this new guitar I've got, man. It'll do this crazy stuff. The new amplifier."

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 the Internet. 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: So you've told us a story of how you came to start your first company. That was Structured Systems Group. Were there any other software companies, like personal computer software companies, at the time? Or were you, like, one of the first?

Cooper: We were definitely one of the first. There were personal computer software companies. They were primarily—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, that had business application software. And that's how he [Kildall] articulated it. He said, "Really, this is the three points of the tripod." Now, there were a bunch of other little, tiny companies doing little system-like stuff, little language-y things, and some other alternative operating systems, and there would be, like, interpreted BASIC operating systems. So you boot up the operating system and it is the language, that kind of stuff. And there were a couple of companies that had made a success selling applications in the mini-computer world, who looked at the new microcomputer world and said, "We can port our software," which was a bad idea. But it was a stopgap enough for them to, eventually, get into the marketplace. So I don't know if I could say that we were the first. I don't think that would be strictly accurate, but we were one of the first and early on. There were also, I guess, some game companies and we, of course, racked our brains trying to figure out what to write. Because it became clear, right away, that there was a real hunger for software, and there were no software distributors at the time, and everything was sold on floppy disks. And 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 little database manager kind of program. And, you know, I got a bunch—there were free little games out there and I got a bunch and spiffed them up and made them a little better, and we sold—for, like, 20 bucks we sold a diskette with games on it.

Hsu: And General Ledger, was that one of the products?

Jon Plutte: Can I interrupt? There's one thing we didn't get is what year was that?

Hsu: Oh, yeah.

Plutte: Yeah, could you give us a preface saying in whatever year it was, there were three major software companies; right?

Cooper: Yeah.

Plutte: Just so we have a context for that.

Cooper: Well, I started Structured Systems Group in 1976. I, actually, started it in 1975, but it was a consulting company, and there were three of us who started this company. We printed out little cards and we went out looking for consulting business, and one of the guys got a job that he could do, I got a job that I could do, and the third guy dropped out and went back and took a full-time job, and so our consulting business ended up with two guys and two contracts. But in 1976 is when we discovered the microcomputer, and when we bought the computer and we came up with this idea of the turnkey system, and built the General Ledger. And 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 in 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 want to—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.

Hsu: And that was 1980?

Cooper: 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, and I turned around and I started a new software company, and I discovered that it's not that easy. 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. But just because you have good software doesn't mean that you have a successful company, and it was, kind of, the company was held up by my force of will, and the largesse of my primary investor, and my primary investor was a guy who had a microcomputer business. 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. And in desperation, I had been negotiating with somebody to buy my spreadsheet, 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 semi-blind 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. And I took...

Hsu: That was 1983?

Cooper: That was—yeah, somewhere around there, late '82, early '83, somewhere there. 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.

Hsu: Okay.

Cooper: And 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 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...

Hsu: This was SuperProject?

Cooper: 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 1980's. 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. And the other program I wrote was, I had been fascinated by the—well, a whole bunch of things interested me. 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 multi-tasking 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. And a friend of mine--

Hsu: What year was this?

Cooper: This was probably '86?

Hsu: Okay. And so weren't there other like 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 inter-process communication, dynamic relocation, and dynamic loadability. 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 inter-process 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, that I couldn't do. So I went home and I put my little graphic front end, and multi-tasking 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.

Plutte: Could you say that?

Cooper: I wanted to replace Windows Explorer with my own, yeah. 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 of mine, who 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.

Hsu: And this was what year?

Cooper: This would have been in probably about '86, maybe '85, but probably '86. 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! Is, this is what the shell needed to be, is 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, an erector set, you know, a Lego block kit, 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, that would let them do what they needed to do. They could start a program that they run with a single button and not have to remember the names of the files and all that stuff. 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. And what I—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 '88—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, who is the founder of Valve. He's one of these characters in the industry who's been very successful. But back then he was just this junior guy. We used to call them the "Gates Clones," because there were all these nerdy looking guys who'd rock in their chair like this, and push their glasses back, and they'd all make sage pronouncements like Bill Gates did, and Gabe was one of them. He was a classic "Gates Clone." And I don't mean this is an insult, Gabe. I mean, he's a brilliant guy. And but it was so interesting, because I started—I went into my little spiel, you know, giving my little demo, and it's a half-hour demo, and about five minutes 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 he's <laughs> he was—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. And so yeah, so he said, "This is--," you know, it took a while there. So I mean, I don't know how many stories you want to hear, but there's a bunch of stories. But we 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. And 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 non-disclosured. So I would go to people and say, "Well, I've got a big--," and they'd say, "What have you been doing?" And I'd say, "Well, I've got this big project. It's going to be released by Microsoft!" They go, "What is it?" "Well, I can't really tell ya." I didn't really know. I mean, it was—anyway, I sounded like I was full of crap. 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. I think it's a—there is one person in this world that's a huge fan of BASIC, and that's Bill Gates. He likes it. He's the guy who invented PEEK and POKE. I mean, BASIC is a language that has no self-restraint. <laughs> It's a cheap date for anyone or anything. And I've never really liked it. It doesn't have the kind of internal consistency of a real language. It's just some random thing where you can, "Oh, you need a function? Let's just create a function. Put it in the language." And even the inventors of the language, Kemeny and Kurtz, fought against that and lost. And 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 pre—"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 that they released it. And it became—it was this incredibly powerful—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 it created an entire marketplace, 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, much, 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 '70s, who had a—who was an author, a technical writer in Marin County in the mid-'70s, 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.

Hsu: Wow, we went through a lot. So there's—I want to go back and touch on a couple of things that we sort of skipped over.

<crew talk>

Hsu: Okay, so we were talking about Visual Basic. Did you want to add anything to that before we move on to the next phase of your career?

Cooper: I'm happy to move on from Visual Basic.

Hsu: So how did you--

Cooper: Visual Basic, I sold that to Microsoft in '88.

Hsu: '88.

Cooper: And that was pretty much the end of it. <laughter>

Hsu: That was the end of your--

Cooper: Well, by the time I delivered—Golden Master was done. It was over and done. It was later in that year, and then it was finally released as VB in '90, I think?

<crew talk>

Cooper: So I'm just saying that the last time I laid a finger on anything that really related to VB was 27 years ago.

Hsu: Yeah, so your involve--

Cooper: But I haven't done a single thing since then!

Hsu: Right.

Cooper: No, no. I've mostly been asleep for 27 years. <laughter>

Hsu: So you sort of shepherded it to its initial release, and then after that you stepped away.

Cooper: I didn't even do that. I mean, I <clears throat>, I shepherded it to delivering at Microsoft, and then they took it to final release. You know? I mean, 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!

Hsu: Yeah, I mean, I was curious, you know, when you were telling us the story about how you started the whole Ruby thing, how much exposure had you had to the Mac, or to Xerox's workstations, or any of the existing graphical interface environments?

Cooper: Well, I--

Hsu: And/or visual programming environments like, NeXT was working on one in the late '80s as well.

Cooper: The—boy, I'm trying to remember when it was. It must have been in the—must have been in about '85 or so, I went to a demonstration at Xerox PARC in Palo Alto, and I saw a demo of the Xerox Star. It was a public demo, so I believe that Gates and Jobs had both been in there before me, and they probably had—were in—on sort of a higher level, and had an earlier disclosure. But I saw the Star, and it blew my mind. And I went home and kind of ripped out a bunch of stuff, and said, "Well, we're doing this differently from now on." 'Cause it really did show it up to be new and different. But the—keep in mind that there was a two-year hiatus between the time—and possibly more, between the time that I sold this dynamically extensible visual programming language to Microsoft and the time that VB shipped. Okay? So in that time, HyperCard came out, and the NeXT became a thing. I never saw any of that stuff. So but this is not a strange phenomenon is that ideas have to come in their time. And I mean, one of the things that I was working on so assiduously in 1990, when I was so frustrated, because I couldn't sell anything to anybody, and I was just—one of the things that I invented is basically it's JSON. Okay? J-S-O-N.

Hsu: Oh, okay, JSON, yeah.

Cooper: I mean, it's—I was so struggling with this notion of file formats that you had to know to interpret, and I said, "That's not okay. There should be at least a way to encapsulate it," and to say, "Here's a thing, if you understand the thing, you can delve into it; and if you don't, you can just skip over it, and you can do so safely." I mean, this was something that UNIX had pioneered, but they had done it a really elemental way, and they hadn't—you know, what they did was they said, "Well, here's a simple—here's a couple of simple things that everybody has to know." Instead of saying, "Here's a way to handle something without knowing it," which JSON does. And so I was trying to build a program that was a kind of a email, calendaring, you know, handle all your data, find everything you need on your hard disk kind of program. And I encountered this problem. There's got to be a way to encapsulate information in such a way so you could hand it around. You know, like giving somebody a book in German, if they don't speak German, they can still handle the book, and give it to somebody else, or put it on their shelf or do something with it. It's not like—I mean, in the world of software, you hand somebody a book in German, their head blows up. That's been the way of operating systems up until just the last few years. So I did this work in '85. And I pitched it, and pitched it, and sold it. I went up and down Sand Hill Road showing it to every single VC I could find. And they all just like, "What are you talking about?!" You know? An idea has to find its time. It has to have all the supporting actors. And—<sighs>

Hsu: Yeah.

Cooper: That's the thing about innovation.

Hsu: Right. That's sort of the—so visual programming, that was something that 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. That's all. Wait. What do I mean that's all? That was awesome.

Hsu: Well, you know, so this is interesting because, you know, you mentioned earlier that, you know, the relocatable, the dynamic relocation and the dynamic loading, those were two critical, technical pieces. Could you maybe explain that a bit more to somebody who might not have the technical background, to explain what those are and how they enabled Visual Basic?

Cooper: Sure. There was a lot of controversy at the time about multitasking, about whether multitasking was preemptive or non-preemptive. And it's—the thing about preemptive multitasking is it means that the operating system takes your authority and control away from you and then gives it to somebody else. In order for that to happen, the environment has to allow your program to go quiescent while another one's active, and it means that somebody in the middle has to know how to apportion resources. Okay? That involves a lot of down in the guts of the operating system knowledge, but if you're not down in the guts of the operating system, you could do multitasking. It's just that you kind of have to do it on a voluntary way so that instead of, you know, this Godlike third party operating system comes in and puts you on ice and fires somebody else up, what happens is your app just tidies everything up itself, and then it says, “Okay, I’m done.” And then the next guy runs. Okay. So if you’re responsible for your own stuff, you can do just fine multitasking. It’s non-preemptive. So I couldn’t get down into the guts of the operating system because it wasn’t my operating system, but I needed to do some multitasking. The idea is you hit a key and it goes and does something, while in the meantime you have to paint the screen. Well, again, if it’s just everybody has to be well behaved, that’s all. It’s no big deal. There’s an enormous brouhaha in the development community about whether non-preemptive multitasking was proper. Okay. And it was, you know, how many angels can dance on the head of a pin kind of a debate, and there were people saying, “I can’t use that. It’s not preemptive multitasking.” Well, doesn’t mean anything. But what is meaningful is when an application gets loaded into memory to run, it’s located in a place. Okay. Well, then the next guy running comes and gets loaded next to it. Okay. Well, this guy finishes. Well, now there’s a hole in memory. Now if a guy comes along that’s this big, wanting to run, there’s half the room is here, and half the room is over here, but there’s this guy in the middle. Okay. So what has to happen for it to be practical is dynamically, in the middle of running, transparently, the operating system has to say, “Stop for a microsecond. Move this guy down. Adjust all of his internal addresses so he’s gonna operate properly, and then let him go again, and load the big guy in here in the empty space.” That’s dynamic relocation. Without dynamic relocation your system chokes about the second or third load. You might as well not have multitasking, preemptive or not. The whole preemptive multitasking was a red herring, and what was important was this ability to dynamically relocate. And like I said, at Digital Research, I kept fighting for it. These guys were all ivory tower engineers. They go, “Why would people want that?” I just told you I wanted it. They go, “Nobody would want that.” So Microsoft got it, and as ugly and wart covered as their operating system was, it did dynamic relocation, and that’s what matters. So that’s why I threw in my lot with Windows. I became a real pioneer in the Windows world building software in that environment, and I wish somebody with more operating system skills had written that, though. I mean they really did literally have summer interns writing their stuff. I looked at their source code. During the builds, we were swapping builds with Microsoft, and they would send us source code, and I saw it. I saw it. It’s code that only a mother could love. They couldn’t figure out how to do— They broke the 640k barrier, but what they couldn’t figure out how to do was to create their call headers to work across the large model boundaries, and one of the guys working for me at the time—Microsoft got it so bad that you had to do source code fixups. So before you called one of their large model subroutines, you had to do an operating call that was essentially a link edit call. You had to say, “Save everything now because I’m gonna do something risky.” And then you’d call across a 32-bit address. And then when it came back you could say, “Okay, I’m good now,” which is pathologically bad, because if you forgot one of those fixup calls, it just meant that your program would go off in the weeds somewhere at some unknown time in the future for some unknown reason. It was an untraceable bug. It made everybody nuts. Everybody hated it. Everybody knew it was crap. Well Mike Geary, the guy who was working with me on the Ruby project, one day just said, “I’ve had enough of this crap.” And he went, and he started looking at it, and he realized that Microsoft’s own compilers, we were using their compilers, their compilers left two bytes unused in the header, and so he went in and he put code in there. So he wrote a program. It was called “Fix DS,” a tiny, little program. If you ran this program in—I’ve forgotten the details of how it works, but you do a compile, and then you do a link edit. Okay. Well, you do a compile then you run Fix DS, which would take about a half a second, and then you’d run the link edit. What Fix DS did is it went around and it found when you had a large call, and it put a fix up value in those spare two bytes that

Microsoft, themselves, had left in there. What it did was it made it all large model safe. So we didn't have to put those source code fix ups in our code. And the Microsoft guys freaked out when they saw it. So Mike ended up flying to Seattle and explaining to Microsoft how he had solved the problem. And they—I think I was with him on this first trip, because I remember them looking at him and saying, "That can't be done." Mike, in his quiet way, just kind of chuckles and says, "No. Let me show you." And I mean, Microsoft were the pros from Dover. We were just this little four-man shop in Silicon Valley. And the next release they didn't have to do the source code fix up, because they used Mike's method. They tried to sue Mike, because after the Ruby project, he was just contracting with me. He went on to do a bunch of other stuff, and he was helping somebody do—Mike was really good. He was really, really good. I don't know what he's doing today. He was doing a bunch of really close to the metal, interesting stuff for somebody. Somebody wanted to do something that Windows really didn't want them to do, and so Mike showed them how to do it, and so Microsoft sued them. So here's what happened is we were doing—Micro—Windows was so buggy we couldn't understand what it was doing under the covers. So we went to them. We said, "Will you give us the source code? We'll bring you the golden master a lot sooner." And they said, "Well." I mean they protested but we made such a case. And they finally said—Windows was in three chunks. There was the kernel, and then there was memory management, and then there was the front end. And so they finally gave us memory management and the front end, but they wouldn't give us the kernel, which was fine, because we didn't really need the kernel, you know, because that was the inner workings of Windows. So we said, "Okay." So they just gave us these two chunks, and it was what we needed, because what we were doing was a lot of Windows and screen stuff, and so that's what we really needed, and we had the source code. So when Mike went off and did this contract, this is long after he left me, but then Microsoft sued him, and he came and he said, "What am I going to do about this," because they said Mike could only have known what he did for this other company because he had seen the source code, and we had to all sign nondisclosures. And so we studied it for a while, and he realized that everything that they were talking about in their lawsuit was in the kernel, which we never saw. We never had source code to that. So Mike had to hire a lawyer, write a letter back to Microsoft and say, "No, no. This is baseless." So Microsoft never apologized to him, never compensated him, never nothing. They just dropped the lawsuit, let it go. That was the kind of relationship that we had with Microsoft. It was not really friendly, you know. I used to say. I don't know if it's true, but I used to say that we were the only company that ever sold core technology to Microsoft that wasn't either hired or acquired, you know, and that is kind of an affront to guys who have their ego tied up in it. It was interesting. One of the make or break things that Gates said to me when we did the deal is he said—I told him I wanted to put my name in the About box of the program, and I said I was getting a lot of flak from his underlings. And he said, "No. You can't do it. He goes, if you insist on putting your name in the About box, I'm walking away from this deal." Nothing else I said. I told him I needed more money. He said, "Okay." I told him I needed more time. He said, "Okay." I said I needed this. I needed—he said, "Fine." I said I needed to get my name in the About box. He said, "No." It was a real shock to me. I had no idea. But he considered it part of Microsoft's culture is that all of the software is from Microsoft and not from individual people. So I managed to actually extract the concession from him—see I negotiated with Bill Gates. I managed to extract the concession from him that in the About box it would say "Cooper Software, Inc." See, that's how Mitch Waite connected me to it, otherwise there would have been no trace of my contribution. Okay? And as it was it didn't just say "Cooper Software, Inc." It said "Portions of this software by Cooper Software, Inc.," and as soon as they did their first point one update, boom, it was gone. So for a few brief moments I had my name in the About box. I mean, I knew I wasn't going to make a lot of money off this project selling it to Microsoft, but I also knew that it was going to be really good PR to sell core technology to Microsoft, and I was in the business at that point of inventing stuff, and moving on and inventing another thing, and selling it, and moving on. And so I had trouble selling a shell to Microsoft's competitors, so I figured, well, good, I'll sell it to Microsoft for whatever I can get for it, at least get from them the bragging rights, and then move on. So we had a clause in the contract that said that they would publicize me, and that I could talk freely about it. And they didn't publicize me, and I started calling myself the Father of Visual Basic. Well, an interesting thing happened is Microsoft started a disinformation campaign about the Father of Visual Basic. It was really funny. They'd have these tech conferences, you know, where people—they'd have lots of presentations, and stuff, on how to code. And they would have—after hours they'd have a get together where you have pizza, and beer, and stuff, and they would run a trivia contest, Windows trivia. And one of the questions was, "Who's the Father of Visual Basic?" And all of my buddies, who were all Windows developers, would all go, "Alan Cooper." And they'd go, "No." And they'd name some guy who was one of their staff developers who worked on the VB project. And all the programmers started calling me up going, "Alan, do you realize what they're saying?" And I just laughed about it. I didn't think anything of it. I just thought that's just Microsoft. That's just the way they are, you know. They just—they'll kick you. They don't care. And then I got a cease and desist letter from their lawyers. They said, "Stop calling yourself the Father of Visual Basic." And so I sent an email to Gates, and I said, "Call your dogs off," you know. "We have an agreement. It's okay for me to do this." And he came back and said, "Well, you know, you really didn't do that much, did you, Alan," you know. It's like, "Thanks, I don't need this," you know. I don't need a billionaire diminishing my contribution. So I finally go, "Look, Bill, I'm not asking for your blessing, you know. Just I have the right. I'm quoting Mitchell Waite. He called me that in a book that you guys supported. It's, you know, this is true. So he backed off. But the guy who instigated this whole thing, the guy who started the disinformation campaign, and the guy who called the lawyers on me, was a senior product manager at Microsoft. Okay. I didn't know him, because he came in from another department long after my deal with Gates was done and I was long out of it. And at this point, VB was a success, and this guy had come in. He was managing a successful product. And I had never met him, but I knew his name. So this exchange I had with Gates, I don't think it was on the phone. I think it was on email. Anyway, after our last email went, a while later the telephone rings, and it's this product manager, who was in charge of VB, and he telephoned me directly, and he was furious. He's just yelling at me on the phone, said, "What the hell is this all about? Why do you keep saying you're the Father of Visual Basic? You don't even work here." You know? And I said, "Well, you want to hear the story?" And he said, "Tell me the story." So I told him the story, and at the end of the story he goes, "You're the Father of Visual Basic." But, see, and this is why he's a senior executive at Microsoft. He goes, "Wait a minute. Would you mind if we honored you?" And I said, "No, not at all." He said, "I'm going to get back to you." And a few months later I got a call from Microsoft. They said, "Would it be okay with you if we flew you and your wife first class to Florida to Windows World Conference, and invited you to a black tie celebratory dinner and gave you an award?" I said, "You know, that would be okay. I, you know, I'm all right with that." And so they did. So we go to the Ritz Carlton in, this is in Atlanta, and it's this huge black tie. There were probably two hundred people in the room. There were probably two hundred waiters in the room. I mean, they really did it up right there. And I got to sit at the big table with Gates, sit right next to him, and we talked through the night, through dinner, and stuff, and then afterwards they had this award ceremony. And I asked one of the factotums. I said, “Am I going to have to talk up there? Do they want me to give a speech to this crowd?” They said, “No, no. You’re not supposed to talk. What you do is you go up and you accept your award. You hand—you shake hands with Bill. They’ll take your picture, then you walk off the stage left.” I said, “Okay, good.” So I go up on the stage, and shake hands, and accept my little trophy. And then Bill points to the podium. And so Gates is standing there with this grin on his face. I know this grin. This is the grin that’s “I just skewered you, didn’t I?” And he starts asking me questions about VB. But he was good. He was good to his word, because he realized. I think he went back and he read the contract and he saw that it was his responsibility to promote me. So he figured, okay, he doesn’t have to promote me on a day-to-day basis, but he promoted me on this one thing. So he gave me this award, and then he turns to me and he says, “One of the controversies was all—the VBX interface was a huge controversy, because one of the idiots at Microsoft had pulled the VBX interface out and said, “We don’t need this,” and threw it away. And Gates came in, and saw that it was gone, and threw a tantrum. And so a rumor went around that Gates invented it on the spot, okay, which was one of the things that the disinformation campaign was using against me. So Gates looked at me, and the first question was, “So, you know, tell us about the dynamic extensibility. How did you come across that idea?” And he just—so I told a little story, and he asked a couple more questions, gave me the opportunity to thank my wife, and that was that. But it was, you know, that was how Microsoft turned a fracas into a marketing opportunity for Microsoft. I got a little pat on the back. I went home okay. That’s how things are done.

Hsu: That’s an amazing story. Before we move on to the next phase, I did want to go back and ask two things, one about your work with Gordon Eubanks and publishing CBASIC. How did that come about? What was the nature of your relationship?

Cooper: Well, in the very, very early days of Structured Systems Group, when we started work on, you know, the idea of let’s build an accounting program, it turned out at the time the only languages that were out there were really toy languages.

Hsu: Like BASIC.

Cooper: Well, they were very, very basic BASICs. Okay. And none of the BASICs had fixed point arithmetic. It all had floating point, very simple, little floating point algorithms that had been cribbed from some university somewhere. And so try to do an accounting program with floating point, you know. It’s just—you get weird, little round off artifacts, and everything. And believe it or not, what we started doing was writing our own—I mean, we didn’t have a compiler, okay, and we weren’t about to write a compiler from scratch, because we were trying to write accounting software. So what we did is—we did have integer math. We have integer and floating point but no fixed decimal. And so what we did is we took two integers, and we slammed them together, and we had like a high range integer and a low range integer to form an integer big enough to do basic accounting. But it was the most kludgy, clunky thing in the universe, and we knew that it stunk up the place. And, oh, God, the story is—it’s complicated. When we—

Hsu: So General Ledger was written in BASIC?

Cooper: It was written in BASIC, but the first product that we did was we got our operating system from Gary Kildall, CP/M. And we were asking him about languages, and he said, "Well, there's this—one of my grad students built a BASIC language," and he gave us a copy because it was public domain, and it was
called 'BASIC-E," E for Eubanks." Okay. And his grad student, Gordon Eubanks, who was the chief engineering officer on a nuclear submarine in the Navy, had done this as his master's thesis for Gary, his advisor. And it was a—unlike other BASICs at the time—BASIC was known to be an interpreted language. It was designed originally as an interpreted language. All the implementations of it were interpreted, and Gordon deviated from this. And BASIC-E was a pcode compiler interpreter. So it compiled source code into an object code for a hypothetical machine, and then a runtime interpreter emulated that hypothetical machine, so it was called a compiler interpreter. The nice thing about a compiler interpreter, it's generally higher quality. It's slower and clunkier to write for, but it gives you source code protection, which was really nice for us in those days. It was a valuable thing. Gates' BASIC at the time was interpreted. But BASIC-E—so that's what we started to use and using the integer math in BASIC-E we started to cobble up this thing so we could build our accounting system. But the problem is BASIC-E had some bugs in it. So we asked Gary if he could put us in touch with this grad student of his who had written it. So he got in touch, and so one day we drove up to Vallejo, which has a big Naval base, and we visited Gordon Eubanks. Gordon, he didn't know who we were at all. We were just a couple of random guys. There were a lot of random guys in those days floating around doing crackpot things, and so we came in, and we asked him a lot of really pointed questions about BASIC-E, and he believed that we were legit enough that he gave us source code, but we weren't legit enough for him to give us soft source code, so he gave us a hard copy printout of the source code in this proprietary language called "PL/M" that nobody had a compiler for unless you owned an Intel development machine, but using this printed source code we came home and reverse engineered enough of BASIC-E so that we could fix a couple of its more egregious bugs. And then we hot patched it, and my partner at the time put his first initial in next to the "E," and so it became "EK." I forget how he did that, or "KE," or something. And then we released it into the wild. Again, because it was public domain then we just put it out there. Well, then we went to the West Coast Computer Fair, and there, at the Digital Research booth, there's Gary Kildall and his wife, Dorothy. And there was Gordon Eubanks, and we introduced ourselves as the guy who had taken his printed source code and patched it. He went, "You're the guys. I've been seeing that K version going around." And so our bona fides were established. So Gordon, at that point, brought us into his confidence. He said, "I've been approached by the IMSAI Company, the same company that made my blue computer. They want me to make a commercial version of the"—boy, you know, it wasn't IMSAI that asked them to do it. It was—what was it? I believe it—maybe it was Radio Shack. Sorry. I'm getting lost in the details. One of those guys asked him to take BASIC-E, which was his public domain little compiler interpreter basic, and create a commercial version that would be proprietary, and would have things like fixed decimal arithmetic in it. And he said, "Yes." He took on the contract, even while he was still active duty. It was interesting, but he started writing this commercial version of his BASIC, which released to the world as CBASIC. And, you know, I was the application guy, right? So Gordon said, "What would a commercial BASIC have in it?" So I had been learning about, you know, structured techniques and stuff, so I was helping him to put some structured constructs in the language, because he

could build most anything. And so we would get together at night, you know, and we'd sit around after he would come home from the Navy and we would figure out what we were going to add to the language to make it a commercial BASIC. And then at one point he was about to ship out. He was in Hawaii, and I ended up flying over there and spending a couple of weeks with him, you know, fixing everything, because, like, he shipped the version off. And the next day he went underwater for six months. Anyway, so we, one of the first products we ever sold, I think we sold, like, the public domain, the patched version of public domain for, like, twenty-five dollars, which was basically our cost. But then we sold CBASIC. We published it. I wrote the manual for it. We found a printer. We printed up five hundred copies of the manual, and we put an ad in Byte Magazine, and so we were the first publishers of Gordon Eubanks' first real commercial product. But, you know, Gordon, he was a submarine driver, and ultimately he decided that the only person he could really trust when he was underwater, because he was literally underwater for six months at a time, was his mom, so he had his mom run the business. So we still sold it, but we didn't have any control over the product. So we had an interesting relationship with Gordon.

Hsu: Could you talk a bit more about your relationship with Gary Kildall? Like, how did you meet, you know. What was that relationship like over the years.

Cooper: Well, we met him at some computer get together somewhere, might have been the Computer Faire in San Francisco. I mean, we were looking for system software to run on—I mean, you buy this computer and then, you know, you could make it turn on but not much more than that, and so you needed an operating system. You needed a language, and Gary was one of our first contacts, and he had an operating system. He had built this operating system, I believe, while he was doing some consulting for Intel. He was still teaching at the Navy post grad school in Monterey, and he was just a brilliant computer scientist. He was very intimidating to me, you know, because I was, you know, a COBOL programmer, really. And here was this guy who had some scientific heft, and he was the guy. His, I guess it was his doctoral dissertation was relocation, how to do relocating fix ups inside eight-bit computers, and it was really seminal work. And he—yeah, I got this operating system. It cost sixty-five dollars, and, you know, on a floppy disk and you could boot it up, and it would come up, and it would say, "A>." [pronounces 'caret'] It was miraculous. You could type "DIR" and it would tell you what files were there. And it had a lot of really annoying characteristics, like if you ever changed a disk on it, it had no way of recognizing that you had changed the disk. I mean it had all kinds of ways to recognize that it changed the disk, but it refused to care. And so if you changed the disk, and then you kept on computing [clap] it would just trash your disk. So what you had to do when you changed the disk is you had to say "control C," which would clear out its mapping of your disk and start over from scratch. Well, this is what I came later to call an ejector seat lever. It's like turn on the radio. Turn on the fog lamps. Flip the ejector seat. Boom, you're out. And that's what it was like. And it was one of the reasons why early microcomputers, they had such a reputation for being hard to use is because of stuff like that. And people go, "Gary, you got to fix that." He goes, "Oh, no, it's really, it's okay. People will do it." And it's one of the reasons why Bill Gates crushed him in the marketplace, is Bill Gates said, "Let's take that out of there," and he did. Yeah. It wasn't much of an improvement, but it was enough. So Gary was an enigmatic and interesting guy. It's my understanding that Gary's—Gary wrote a book, a memoir, that is, I believe, is being published as we speak. Gary died in mysterious circumstances at age 52 in Monterey. He was a wealthy man, very successful by all measures, except one, and he had a beautiful wife, and happy kids. His son is a real prominent artist, but it really, really bothered Gary, I think, that his alma mater, U-Dub [University of Washington], honored Bill Gates and not Gary Kildall. On the other hand, the university's a business, and Gary, you know, probably didn't endow a new building the way Gates did. What are you going to do? Luckily, all that stuff was over my pay grade. But it was fun working with Gary because he's, I mean, he was, he's probably one of the greatest programmers to ever walk the earth.

Hsu: Great. Could you talk more about what it was like to work with him?

Cooper: Hmmm. Oh, God, it's been so long. No. Ask me another question.

Hsu: Okay. <laughs> Do you want to tell that story about the executive and the Visual Basic prototype?

Cooper: Well, I mean, I really pissed off so many people at Microsoft. I just had a real knack.

Hsu: <laughs>

Cooper: And so if you read Fred Brooks, you know that what you have to do is throw one away. It's going to take you at least a couple times to figure out how to do whatever it is you're doing, and I believe that. You know, and I still believe it today. I think that Fred Brooks is exactly right. And I think that one of the banes of my existence in the '80s and the '90s was people who—how does it go? Is it--

Hsu: So you write a thing, you write your prototype, then you throw it away and you redesign and you start over from scratch to write the real version?

Cooper: Yeah. Yeah. That's what you do, is you write it to learn what it is you're doing and how to do it, and then you throw it out and then you do it again. Because the most important thing you get out of writing a program is knowing how and what you're writing, not the code. The code itself is just a big pile of technical debt. So, and this, of course, was known long before the term technical debt existed. So I had sold to Bill Gates. I had demoed and sold to him a prototype. It was probably, I don't know, 15 or 20,000 lines of C code, and it was some pretty hacky, ugly stuff. But it was all proof-of-concept and this is how it's going to work, and so what I did was I took that and what I knew in my head about what I wanted, and I wrote a big, fat, technical specification describing all the functions, all the features and all the interface and the characteristics of it, and then I chucked the prototype out and we started working from scratch. Building our libraries and everything properly, and when Gates inked the deal with me he then turned it over to this, to this guy, this manager who's in charge of the Windows 3.0 release and I told him that I threw Ruby out, and he lost it. He went ballistic and he started screaming and yelling. He goes, "How can you do that? You're going to make me late. I'm not going to make my deadline, and you're screwing everything up. All of that perfectly good code. Why--" you know. I just said, "Ah, it's gone." <laughs> I mean, I don't know, but… So of course, throwing it out was the smartest thing we did because we were able to build a nice, orderly system and we delivered to him on time, and that was fine; and he shipped eight months late. Because he didn't throw out his prototypes, you know, and that's one of the lessons to learn, is that, is that software development is filled with counterintuitive truisms, and that's just one of them, and that's what experience in the software business is. It's worth more than any, any book learnin'.

Hsu: <laughs>

Cooper: No. Somebody told me this the other day. They said, they said, "Software's a business where the technicians who know nothing about business are, work with business people, who know nothing about <laughs> technical stuff." or something like that, and it's kind of true. I mean, that's, that's the world we're in.

Hsu: <laughs> Thank you.

Cooper: Or as somebody once said, it's, you know, science is giants standing on the—it was Leibnitz or somebody who said, "It's giants standing on the shoulders of other giants," and people have said that software's midgets standing on the toes of other midgets.

<laughter>

Cooper: I mean, if you can't laugh at what you're doing, somebody else is laughing at you. That's the way I look at it.

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

Cooper: That is presentism and historical revisionism.

Hsu: Okay.

Cooper: And no, it's not true at all.

Hsu: Aha.

Cooper: And it was considered to be a huge step down and backwards and a step into irrelevance, and in fact, it wasn't the journey that I made. Because after I shipped Ruby to Microsoft, I went off and invented another product, and I learned, I learned—keep in mind that this is in the late '80s, early '90s, and the industry, which, you know, the pendulum swings between innovation and consolidation and innovation and consolidation, and it was going through a consolidating phase, and I learned a new word, monopsony.

Hsu: Yeah.

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 publishers, my, 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.

Hsu: The accounting program?

Cooper: Yeah. 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, so… And I learned so much building that General Ledger that when I then went on and did an accounts receivable, accounts payable and a payroll and a 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 a software—an inventor and a developer, but that began my path of 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. Okay, and 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 afterwards. 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 said, "Well, who--" 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 kind of 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, and they did. They hired me because they knew me as a programmer, but 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, that was 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 by, I mean, 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 '92 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. We hired—I mean, keep in mind, in 1994, there were no university programs on the face of the planet that taught this. The name Interaction Design barely existed. Okay? There were no jobs anywhere, in any company, called User Interface Design, Interaction Design, Design, nothing. Okay. 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 '90s, 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 guys and 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. There were other companies, I think, who were doing this. There were companies who, like IDEO, was really an industrial design firm who more and more of their clients were saying, "But what about this?" the buttonology, you know. "You did the sleek thing but what about the buttons?" and so they were nibbling at this but we were the only company who was pure. This is what we did, is this interaction design.

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, but they're not. I mean, you think of it as a tree of all these things coming together, but it's not like that. Instead 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…

Cooper: The main focus that people had was on—there were different ways of understanding what had been built. How do you analyze what had been built? But because I was the only guy who was coming at it from a point of view of having to answer these questions before I built it, I didn't see it that way, and the number one dissention that I had in the world was with visual designers and with academics, guys, HCI people, who did this kind of analysis. They would say, "But how can you know, how can you know what's going to be good design, until after it's built and you've put it in front of people and empirically studied it?" And I kind of, "Well, you know, you can't prove it, but you can know," and so I articulated what seemed to me to be fairly obvious stuff, which is make the software be well-behaved. It's not about examining the functions and making sure that the function set is complete as much as it is, is that people want to be able—they're used to looking at a machine and seeing that the lever is thrown this way and the gears are turning. Okay. They may not understand the gears, but they know not to stick their finger in and they know that something's happening, and they know that if they watch those gears go around for a while and nothing comes out, maybe it's not happening right. It is giving clues as to its state, okay? Software, you'd push a button and it would just go silent. It didn't give you any clues about its state. You know the ubiquitous progress meter? Didn't exist back then. No such thing. Okay. So it's constantly, software was just going away, and then it would come back and it'd say, "Do you want A or B?" It wouldn't say, "If you have A, this happens, and if you have B, this happens." It just would say, "A or B?" and if you were wrong, too bad for you. Things would go badly, and so I just looked at this and say some of the fundamentals of software design are, it needs to be revealing of state and progress. 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 rasa, 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. I remember I was giving a class in software design through the extension. This was probably '90, 1990 or '91, somewhere around there, and I was working on some of my ideas. But in the class I made the assertion early in the course, I said, "There should be no error messages ever from a piece of software." This is, saying that in this class of software developers and the people who love them, it was like saying that babies should be killed at birth. I mean, what I was saying was so frightening to them that they got angry at me and they complained to the administration that I was speaking heresy and… But back in the day, you used to get error messages all the time. It would say things like, "Context error." What the hell is that, and what do I do? Then it would say, "Okay." You know, this, this is just wrong, and it used to get errors all the time, and the error messages would, I would call it stopping the proceedings with idiocy, because it would just come up and it would say things like, "I just imported your file." You'd say, "Import the file," and it would put up a modal dialogue box saying, "I just imported your file." It's like, don't brag to me. I don't need you as a software to do that. I mean, it's okay if a five-year-old kid says, "Mommy, mommy, look at me," you know. But it's not okay if software does that, and I said, "Look. If you want your software to be regarded as polite and well-behaved, make it polite and well-behaved," and polite doesn't mean that it says, "please" and "thank you." Polite means that it says, "How are you doing? I'm interested in you. I'd like to know more. Is this the kind of—last time you liked this. Does that mean you want the same thing this time or something—how could I help you?" This is the way software should behave. Well, this is just, this is crazy stuff, you know, 28 years ago. People didn't, they didn't believe it. So 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. Everything about software is a functional description of functionality. So you write your procedures and about actions that the computer takes, that performs functions, that are, have utility for the user. Okay. 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 functions, and inside the organizations that create it you'll find those spreadsheets with a list of functions. You know, the burndown chart. 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. One is the thing you do and the other is the thing you want and they're two dramatically different things and two dramatically different ways of framing the way you think about design, and it's one of the keys to understanding, to answering the question, but how can you know? Well, you can know by saying, not "What are you doing?" but, "Where do you want to be and why do you want to be there?" and so I give an example. Like in 1850 if you were in St. Louis and you wanted to be in San Francisco, you packed your rifle in your covered wagon and you did it for safety and convenience and security, and in, you know, 1990, if you want to get from St. Louis to San Francisco the one thing you didn't do is bring your rifle for safety and security and convenience, okay? So your goals are identical, but your tasks are completely opposite. That's the thing. Is that 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, and it's just, 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 vouch-safed with engineers and it was a, 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 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 and, and so I ended up writing a book called, "The Inmates are Running the Asylum."

Hsu: <laughs>

Cooper: Talking about this. Saying, you know, it’s not that—and a lot of people thought that I was somehow insulting developers, and I’m kind of going, “Hey, developer here. I’ve been developing for a lot longer than I’ve been designing,” and really it was more autobiographical than anything else, and I was just 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, and I realized that, you know, as I like to say, you know, that you can walk into any company and you can take their top programmer and they could probably keep the company’s books too. They could probably do it pretty easily and probably do just as good, if not a better job, than the bookkeeper does. But you wouldn’t do that, would you? Why would you do that? Why would you take your top programmer, that it’s really hard to find and you’re probably paying him a couple hundred thou a year if not more, and have him or her do the work of your bookkeeper who’s making a quarter of that money and who’s easy to replace?” You just wouldn’t do it, and it’s the same kind of thing. Why would you take your—and programmers somehow take it as a affront. You know, they say, “Well, I can design from the user’s point of view,” and it’s like you’re insulting them to say, “Don’t do it.” 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, 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?”

Hsu: <laughs>

Cooper: 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 ‘90s. 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.

<paused in thought>

Cooper: 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 don't say, “Well, do you want this button here or there?” You don't say, “Well, tell me what you like about our product.” You don't say, “What features would you add to make our product better?” You don't want to make the users the designer. The users are not the designer. The users are the users and just because they—it’s like saying, “Okay, Mr. Taxi Driver, tell me how to redesign this gasoline engine.” It doesn’t work that way, so—but you have to go out into the field and you listen, and you ask really broad, open-ended questions, and then you listen and you listen and people will slowly reveal themselves and they reveal—you go to them and say, “Well, do you like this?” You know, if you’re working for Oracle and 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. And so there’s some fundamental understanding here, which is, you can’t learn much through quantitative measures, or I should say—there’s a lot you can learn from quantitative measures. I don't mean to slam that. What I’m saying in this kind of thing, for interaction design, for understanding what your product should do and how it should behave, 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. What you have to do is, you have to listen to what they're saying and what we began to realize is that if you interview 10 people, what you're going to find is that there's a couple of outliers that are just idiosyncratic and useless and you chuck them, and then you look at it and everybody there has a little of this and a little of that and what you do is, you look for the common patterns. And what you find usually is one or two, sometimes more, but one or two commonalities. Now what you need is a way to take those common things which is—if you design for those commonalities, you've satisfied the needs of a plurality of users, so now you need a way to encapsulate and contain those commonalities and use them effectively in your design work as you're saying, "Okay, now what does this mean to us and how do we satisfy these needs?" And what I—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, you can use this person to define who the user is and what they're trying to accomplish and why they're trying to accomplish it, and it's immensely powerful. 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 screens and here's the buttonology." Then what you do is, you take the persona and you play act, roleplay 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. Now, personas are widely misunderstood in the industry and they're widely abused, and so every few months somebody else writes a blog post saying, "Personas suck. I've got the solution here," and the thing that they're describing that sucks is not a persona and the thing that—their solution that works pretty good is—always seems to be a persona, and a lot of this has to do with the facts that some of the early work of other people who—I mean, I wrote about personas for the first time in *The Inmates* and—in 1999, and I wrote just a single chapter. 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 and—so we're not doctrinaire about that. 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 '90s, 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.”

<paused in thought>

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 now goes by Lane Goldstone, one of our first—I would say she was the—our original synthesizer. 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: Thank you. So this is my last question and I think this is maybe a good way to summarize our discussion, but at the very 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: Well, first off, I don’t--

<paused in thought>

Cooper: I'd be Bill Gates if I could. When I first started out, like I say, I wanted 50,000 dollars 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: Well...

<paused in thought>

Cooper: I just think that you—you know, 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. When you get to be a billionaire, you get lots of good pies to mess around with, so—but on the other hand, if you try to be a billionaire and you hit the mid level, then you're just unhappy. If you get to the point to where you realize that you've been making these decisions so that you get to do what you like, then you go, "Well, I got to do what I like." There just aren't that many Bill Gates in the world and so this is the thing is that Gary Kildall, he fell into that trap. He compared himself to Bill Gates. And I don't care who you are. You could be the finest businessman around. When you compare yourself to Bill Gates, you're second rate. So don't fall into that trap. Like I say, Gary—by any standard, Gary was an amazing guy with a successful life and a successful career and a great family and friends and all the toys he wanted. I mean, he was—he owned racecars and yachts and airplanes and stuff, but he was just eaten inside by that resentment. And I—you know, you're in that footrace and somebody passes you on the final stretch. You can't look at them and go, "That should've been mine," because then you're just going to be miserable. You go, "I got beat." So I—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: Well, is there anything I'd like to add? Let's see. Yeah. There's one thing I'd like to add, and that's that it's...

<paused in thought>

Cooper: There's—we've reached a new level of responsibility as technologists where—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 dollars 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. That's amazing.

END OF THE INTERVIEW