Oral History of Steve Naroff

Part 1 (session 1 and 2) of 2

Interviewed by:
Hansen Hsu

Recorded October 8, 2018
Mountain View, CA

CHM Reference number: X8800.2019

Hsu: The date is October 8th, 2018. I am Hansen Hsu, curator at the Center for Software History at the Computer History Museum and I'm here with Steve Naroff. Welcome.

Naroff: Great to be here.

Hsu: Starting out, where and when were you born?

Naroff: So I was actually born in March of 1961 in Poughkeepsie, New York. My parents were there briefly; soon after I was born we actually moved to the Bronx in New York where I spent the first six, seven years of life.

Hsu: Oh, wow. What's your family's background ethnically?

Naroff: So my dad-- on my dad's side of the family his dad migrated from Odessa, Russia, so there's on my dad's side Russian descent and on my mom's side Hungary, Austria so-- and I guess in terms of sort of our heritage we consider ourselves Ashkenazi Jews so--

Hsu: What did your parents do?

Naroff: My dad was a professional musician and then educator. He toured with the Navy Band for four years during the Korean War and then after that received a master's degree from Manhattan School of Music and plays tenor sax, mainly jazz, but is also a fabulous conductor and has written a book on jazz improvisation in the '80s that was used by many universities so he's just a heavyweight in the jazz community and is 85 and still plays with groups in Florida so it's pretty amazing. And he was my music teacher so I started playing the clarinet at a young age and did a lot of work with him on music.

Hsu: Right, which became influential later on, yeah. And your mother stayed at home or--

Naroff: She was a homemaker; she also did some miscellaneous administrative jobs for companies in the area. She was always busy but my father was the main breadwinner so to speak in the family and he also had a big private following of students so he taught at school during the day and then at night he had kids coming to the house for private lessons quite often.

Hsu: Was your family religious?

Naroff: Not really. We had very religious grandparents and great-grandparents because they were more first generation and I was bar mitzvah'd when I was 13 basically to make the grandparents happy, but religion wasn't a big part of my life. We would do some of the holidays and observe traditions but overall I wouldn't consider ourselves religious but learning enough Hebrew to actually do the bar mitzvah was definitely stressful. <laughs> It was a reform temple. It's interesting. The area we grew up is now inhabited by very religious Jews, Hassidic Jews, that work in New York City. The town we actually settled in was a place called Spring Valley, which is a very nice suburb about 30 miles outside New York City, and it's quite a religious area now.

Hsu: Wow. What was the community like, the area you grew up in?

Naroff: Well, it was-- from the-- when compared with the Bronx where we were in an apartment living-- we were very close to the subway; the whole apartment would shake when it would drive by-- it was very quiet, very suburban, a big lot certainly by California standards. I think we had an acre of land and I would work on the land with my dad and a big pool, just a beautiful suburb, and the schools were exceptional there, which was one of the reasons my dad moved us away from the city and out to the suburbs, which was very common in the day. And for us it wasn't just about him wanting better schools for his kids but it was him wanting a better place to work as an educator, that the schools in New York were rough. My dad actually-- I guess I should go back and mention he grew up on East 105th Street, which is Harlem in New York; he grew up in a very tough part of New York and music-- aside from him being drawn to it and loving it music was a way out of the tough projects and also a way to get respect from all the people there so-- because it was a really rough place.

Hsu: Right. Yeah. Wow. Do you have any siblings?

Naroff: Sister, Debbie. She still lives in Ramsey, New Jersey, which is close to where we grew up, and her husband-- she actually went to the same school as I did and she met her husband there like I met my wife there so we both attended Oswego, which I'm sure we'll talk about later, and we both met our spouses there.

Hsu: Any formative experiences as a child?

Naroff: Well, because a clarinet was put in my mouth at the age of six <laughs> and my father was somewhat of a disciplinarian when it came to music and education most of it centered around music but I played baseball, I was a pitcher, did other things, but-- yeah, but most of it was music and early on most of it was classical music. So by the time I was in ninth grade I was playing the Mozart Clarinet Concerto, which was considered one of the more difficult pieces for the instrument, being graded annually at-- the New York-- NYSSMA would judge you and write up all kinds of critique on your playing. And so yeah, by nine-- ninth grade I was a pretty established classical clarinetist, yeah, and around-- transitioning from ninth to tenth grade I decided that classical music wasn't really my calling; it was very highly structured and it didn't provide that much freedom that I was looking for so I switched over to tenor sax and gravitated much more to jazz and that carried on for the rest of my college. I pretty much stopped playing clarinet and just focused on sax and jazz and something I enjoyed more, but the clarinet was a great predecessor to the sax; they're very similar instruments.

Hsu: Right. Yeah. What kinds of books did you read?

Naroff: Because the music was at such a young age I was listening and reading music and just everything music, and when it came to reading I read the stuff that the teachers required in school <laughs> but when I was looking to just unwind it was much more around music and even magazines. At the time Mad magazine was a huge thing growing up or National Lampoon so when it came to just having fun it was music and magazines and not so much reading serious literature, which is interesting because my daughter actually at a very young age I remember picked up "Lord of the Rings" and read it all. I'm like

“Wow. Where’d that come from?” <laughs> Yeah, and even my daughter actually had an affinity for the “Merck Medical Manual” and would read that also, yeah. <laughs> Once my daughter was in a bookstore with my grandparents and I remember them asking, “Well, what books do you want us to buy?” and she said, “the Merck Medical Manual’” and-- yeah, so there- there’s reading somewhere in there but as far as I go it was heavily tilted towards music and when I got to college I-- and fell in love with computers it was a different story. I was reading everything I could about computers and in those days, as we’ll talk about, it’s actually hard to find the books. We actually had to go into New York City in- into some really obscure places; it wasn’t like going on Amazon and searching <laughs> for stuff so—

Hsu: Right, but that was primarily nonfiction about computers, not science fiction or anything?

Naroff: Exactly. I mean I was into science fiction TV; I was a big “Star Trek” fan back in the day so I tended to watch series or “Twilight Zone,” into that type of stuff but not so much reading, more of a visual person.

Hsu: Talk about education at school.

Naroff: Now where do you want me to start?

Hsu: K through 12, high school.

Naroff: Okay. Well, I was a good student. I was always meticulous in my notes. In fact, I actually brought some notes from my compiler class we could browse later from college. I was always a meticulous note taker and school came pretty easy to me. I wasn’t ultra competitive when it came to school and regarding SATs later on and tests. I was not a huge believer in that type of testing but I did well enough to get around and to do what I needed to do. In high school, I did well in math but I didn’t-- it wasn’t being taught in a way that excited me about well, what can you do with this stuff; it’s like okay, I’m doing well in it but what does this mean? So I didn’t really bind with it, which is why I kept on getting back to music, but math and music have interesting relationships and so I think maybe the music background made it easier for me to grok some of the math. The school was in Spring Valley at the time-- I don’t know what it’s like now but at the time was very highly rated, very well supported by the community and very diverse-- a very diverse group. In that area, unlike this area when I say “diverse” we didn’t have a lot of Indian and Asian people in that area but whatever people we had it was diverse, had a lot of African American folks. It wasn’t just rich white folk <laughs> in the school; it was a broad spectrum, which I actually feel is great. I think that learning how to interact in a public-school setting, assuming the public school is a good school, is the best thing. I tried to keep my kids as much as possible in public schools. In high school, we ended up deviating from that and sent them away to a boarding school but that was later on in life, but as far as my schooling it was fairly uneventful. I’d say the one thing that’s worth noting is my dad taught in the same school-- high school and junior-high school I went to so he was my music director and that was challenging because he’s a disciplinarian; he was the head of the orchestra, the conductor and really kept people in line. And at that age kids didn’t understand discipline as much, I mean some kids did, so I always felt like kids if they got yelled at in orchestra after orchestra they’d look at me like “Man, your dad’s grumpy” <laughs> so it was mixed. And my dad also because he was there was able to keep an eye on me and if I was out of line and another teacher talked to him it wasn’t a fun experience after school, but

we didn't have that many of those instances. I was a pretty straight-ahead kid and did what I was supposed to, not very rebellious, minor stuff surrounding music but not too bad.

Hsu: Good. Did you have any mentors, favorite teachers, heroes growing up?

Naroff: Most of my mentors and I don't know if "heroes" is the right word but people I respected were professional musicians who I was going to see in Greenwich Village in jazz clubs like Sonny Rollins or Michael Brecker who were the saxophone players of the day. I really looked up to them because I was a talented musician but I didn't have the dexterity or the work ethic that some of these other people that attain stardom do; most of them work five, six hours a day on their instrument and I had natural talent and I might play one or two hours a day and I understood the difficulty in getting to the-- to where they were and making it look as easy as they made it look. So I have a lot of respect for musicians but in terms of academics in the day I ended up having most of my computer science mentors-- because of the year because I graduated high school in '79 computers were really in their infancy as far as-- there were mainframe computers but certainly people like Steve Jobs and others weren't around to be revered at that point but in college-- we'll talk later-- I mean there's lots of interesting people I ended up having a lot of respect for and considering them mentors even if I didn't know them. In college I felt Alan Kay was a mentor just from reading what he was writing. Yeah.

Hsu: Well, that's a good segue to talk about when, where was your first exposure to computers.

Naroff: It was college. When we were deciding what I was going to do after high school and for a while I thought I just wanted to go music-- just become a musician. My dad sat me down and talked some sense into me and even though he was a supporter-- a big supporter of my music since he had toured the world and knows what it's like to do music as a career he suggested that I minor in music, continue doing my music but consider another major, and the major that we casually talked about at the time was computer science because computer science-- undergraduate computer science curriculums were not even that common in the day. In fact, I did a little research to lead up to this interview and I didn't realize it but the computer science curriculum at Oswego where I ended up was founded in 1970 by Oebele Van Dyk who was a Danish mathematician who found his way to Oswego and in '68 wrote a proposal for having a computer science curriculum. And I guess Potsdam was the only other state university of New York that was considering computer science undergrad, basically bachelor of arts degrees in computer science, and there were 60 other SUNY schools with bigger names like Stony Brook and Albany and Binghamton, bigger hubs, and they didn't have computer science curriculums at the time. So I guess another note on this as well is when I was doing research [for this interview] I wondered well, when did Stanford University start an undergraduate program, and I was actually shocked that Stanford did not have an undergraduate curriculum for computer science until 1986; they had graduate programs but no undergrad. So I didn't realize it at the time but it wasn't an easy domain to find schools to educate you and so yeah, we-- well--

Hsu: So is that why you went to SUNY Oswego because they had a computer science program undergrad?

Naroff: Well, I wound up at Oswego for several reasons. Number one, it's worth noting that because my dad was a schoolteacher and because we had to consider the reality of what it would cost and we were

fairly conservative people and didn't want to take out massive loans we pretty much said the SUNY schools in New York State were fabulous at the time, why not just limit our search to SUNY, which we did, and then once I started looking at the SUNY schools Potsdam and Oswego were the only places with computer science programs of the day and they also were receptive to me being a music minor. See, that's the other trick. The interesting thing is when a school adds a degree program like that sometimes they don't want you to minor in something that's as demanding as music; they just don't want you to be distracted. So Oswego from the get-go was like "Yeah, we'd love to have you play in our orchestra and jazz band as a minor and major in computer science" and Potsdam was not quite as receptive to that so I think we decided on Oswego for that reason; it was also on Lake Ontario, which was just beautiful. The only downside in terms of locale with Oswego is it was really cold, so cold and so windy that if you were under a certain weight you could not walk outside near the lake for fear of being blown in <laughs> to the lake so yeah, it was-- I think they have tunnels now or some of the schools have tunnels but in the summer it's just a gorgeous place. So just to recap, it was a good value. I think at the time it was roughly $6000 a year; now it's 27,000 if you include room and board, which you have to include because no one lives near Oswego <laughs> so-- And I had a modest loan, I think I had a $3000 school loan which was paid off pretty quickly, so that was a reason. And as we'll talk it ended up being-- I guess you-- do you want to hear a little bit more about Oswego?

Hsu: Yeah.

Naroff: Yeah. I mean it ended up being a perfect place for me because a fellow named Craig Graci, who was my mentor there and taught me most of my computer courses, had a master's in math and he had decided to start teaching computer science in 1978, which was the year before I showed up, and so he was learning about compilers and operating systems and all of these great things at the same time we were and he was a great teacher but it makes you even greater when you're learning it along with the kids. And he was just an amazing person, was a big part of my life there, also very into music so we were-- a kinship there as well and just took a lot of personal interest in me, which-- considering I wasn't a crazy-great student in the sense that I wasn't-- I was never really pushed hard in high school, I just sort of naturally got by and did well and had pretty good grades, but in college, even a college like Oswego where you're studying compilers and operating systems it was really challenging, it was not easy, and having him take a personal interest in me made all the difference. Another person that taught that came about two years after I showed up, I think around 1980, was Doug Lea; Doug Lea was also a mentor and Doug is-- Doug and Craig are still at Oswego. And Doug's a very prolific author. He's written books on Java, concurrency, C++, he's like a luminary in both the C++ and Java communities, and both those guys just love teaching. They never felt any need to get into industry because when I got into industry I called Craig and I said, "You know, there's so many cool things to do in industry. Are you interested in getting out of academia?" but had no interest in it, just purely-- and he's the director of their Cognitive Studies there now and yeah, it ended up being a great place. When I left Oswego even though we did all our work in punch cards I printed my compiler and still have it to this day and it's something I'm really proud of; it was done in PASCAL.

Hsu: So the first program you wrote was part of one of your classes?

Naroff: Exactly, and I don't remember what that program is or was. I mean obviously we all do "Hello, World!" in whatever language we're learning. I do know my first course was in FORTRAN; FORTRAN was the language that was supported on the IBM 1130, which was a big mainframe, and I'm sure they were just probably basic sorts or whatever you do in FORTRAN. <laughs> FORTRAN, I think I only took one or two courses where that was the language. I programmed in lots of other languages. We programmed I think my second year in a language called GPSS, General Purpose Simulation System; it's a statistical queuing theory and sort of Joe's Barbershop type of problems, and that was an interesting course. And then most of my courses were in PASCAL. Yeah.

Hsu: And mostly on mainframes as well, punch cards as you mentioned?

Naroff: Exclusively. The IBM 1130 had punch cards. We would do assemblers and compilers with decks that were like this big, sometimes multiple decks, and you had three runs a day to get your program right and that required a lot of discipline. I mean we used to flow-chart everything and used to really think about our problems before giving it to the computer and it's interesting. I don't want to sound like an old codger, but even though interactive development environments like the ones we ended up developing at Apple and NeXT and so on are wonderful. There's also something wonderful about thinking about your problem that much and having that level of respect for the <laughs> computer, right, and it took a lot of discipline to get a compiler, a big piece of code, thousands of lines to work under that constraint so yeah, it was time-sharing. Later on the Burroughs 5600 I think-- they added a Burroughs computer but it was-- all the work was on those two mainframes.

Hsu: And did you first discover object-oriented programming while you were at college?

Naroff: No. The closest thing that I remember in college is talking about Simula to Craig and I asked whether he was going to end up teaching a course in Simula and he said, "No, not yet," but there was definitely a respect for Simula that was being done in Europe but no, we didn't do anything with it, and to be honest most of those days were spent thinking of modular programming, not using go-tos and stuff like that, and programming in the large and the benefits of programming in the large that are where objects start reaping benefits that wasn't really a focus so much. In fact, even in my first job, which we'll get into, objects weren't a part of their methodology and frankly it took a while for objects to start catching on.

Hsu: Right, but you were doing structured programming and modular programming.

Naroff: Very structured, very modular. I considered the programs I wrote to be art and took great pride in even indentation, which again is sort of odd because you're never really looking at it, right, but when you printed it you had to look at so it was really important to just manage complexity and certainly indentation, making sure you didn't have routines that were unwieldy were all part of trying to manage the complexity, which-- without managing the complexity you wouldn't be able to get [a] complex program to run in three runs a day. So I can't sit here and actually say the art-- that I was doing the art for art's sake; it was more a side effect of how do you really make something this-- a puzzle that's this complex work under these constraints. And so that's part of the reason I guess I feel it- it's-- that part of the rigor and art is missing today and encourages a lot of sloppy work, sloppy programming.

Hsu: Did you have any summer jobs or internships?

Naroff: Before my junior year, I did odd jobs like cook at Nathan's, which was a hot dog place in New York, but my junior year I decided to apply for an internship at IBM and I got that job as a computer operator and that job was basically baby-sitting these really expensive high-speed laser printers, keeping them fed with paper and data, and these laser printers were just a spectacle. I mean the-- you couldn't even see the paper fly by and this was in-- what, let's see-- basically around '80-- 1980, 1981. Probably. Yeah.

Hsu: I think one of the documents you showed me said 1982--

Naroff: Eighty-two. Okay. There we go. Great. So yeah, I did that internship with IBM. It was local; it was in a place called Franklin Lakes, New Jersey, and Franklin Lakes ended up being across the street from my first real job out of school; it's just kind of interesting how that worked out because that part of New Jersey was not a hub of technology but there were some interesting places to work. And so yeah, the job paid well I think back in the day-- I was being paid $14 an hour if I recall, which for-- if you do the math it's probably like 30, $40 an hour by today's standards-- and it was just-- I didn't learn much but I got to see the monstrosities that <laughs> basically print lots of stuff. Hopefully, there's a lot less printing that goes on today-- I would imagine there is because we have such great storage capabilities-- but yeah, that was a great internship.

Hsu: So let's get to your first job. So you graduated in 1983?

Naroff: Graduated in '83 and we didn't have LinkedIn in those days. I forget how I arrived at the two job interviews I ended up taking. Well, I-- actually I remember one and the other one opportunity I didn't take. I flew from where I grew up in New York to Pomona, California, to take an interview with General Dynamics. I totally forget how I found them or they found me; chances are they found me because I wasn't looking for them. <laughs> And so they said, "Hey, we want to talk to you" so I flew out-- I had no idea what I'd be working on and I flew out and I'm sitting in the waiting room and they give me these brochures on the products that I'd be working on and they were missiles, that I'd be working on FORTRAN code for their computer-aided missile program. And right then I guess I should have realized that may be the case but that was culture shock to me; I really didn't want to work on war instruments. So I took the interview; I have no recollection of it but just sort of tuned it out but it was good experience to do-- it was my first job interview so it was good experience. It was also I think the first time I was ever on a plane. My kids grew up in a different world and they've been flying since they were little kids but yeah, I had no opportunities to go on a plane so that was a little stressful I guess at the time, I'm flying by myself to-- but the other job I do remember what happened. My mother actually said her friend Rosalie down the road works at a cool little company called Henckels Haas & Brown and I said, "Okay" and she contacted Rosalie and said, "My son's graduating with a computer science degree. Would you be interested in talking to him?" And I got her a resume and they called me down and they were-- I don't know-- maybe ten miles from where I grew up so very close and probably within two or three miles of my internship at IBM and ended up loving the place, a great place. They were actually a company founded by three MIT graduates, Lutz Henckels, Ken Brown and Rene Haas, and they were pioneers in the circuit simulation, test simulation business. They had written some really interesting software that they ended up calling

CADAT, Computer-Aided Design and Test, and like a lot of startups as hardware folks they along with some early employees built a product and like many startups realized they needed to then reengineer parts of it, extend parts of it. They were growing out the customer base so there was lots of stuff to do and so they hired a pretty nice crop of young folks out of college; I'd say between five and ten of us started the same time. I'm still friends with a lot of the folks, Columbia graduates-- Columbia University graduates, Carnegie Mellon, MIT, I was probably from the lesser-named schools, but it all worked out great, learned a lot; it was a great culture. The two things I remember working on there was they had a binary circuit file which was-- they had a language for describing how the circuits were connected and that would compile into a binary file and they didn't have great tools for looking at the binary files or analyzing them so I worked on that and later on when sockets were added to UNIX-- actually it's important to note they were a UNIX and a VMS shop pretty much. When sockets were added I worked on some software to distribute across different machines, passes of circuit simulation to speed up the process, and that was kind of bleeding edge at the time because sockets were brand new. The whole notion of using the network and other computers on the network to speed up circuit simulation was considered pretty cool so that was a fun project.

Hsu: So was that your first introduction to the UNIX world?

Naroff: It was. First introduction to UNIX and—

Hsu: Writing a lot of C?

Naroff: --and C and it was demanding. I mean that job was demanding mainly to bootstrap my knowledge of all those things. The work itself seemed-- once you had the tools under your belt whether it be the C shell or learning make or any of these tools-- there's just so many tools you had to master to get your job done but once that learning curve was over it seemed a lot easier. And I forget how long it took; it probably took I'd say three to six months to feel comfortable. I was there for a couple years but once I felt comfortable life was pretty good and it was a fun place to be. The management was very cool; it was a great culture, just a great first job. C was tricky. I mean I should note that I was a little bit surprised coming from PASCAL which encouraged safety to be working in a language like C that was kind of neutral or encouraged bad things-- I won't say encouraged bad things but it was very terse, right. It was easy to make mistakes-- very easy to make mistakes and that's one of the reasons I ended up thinking about-- that's when I first started thinking about isn't there a better way. C's cool but fundamentally it was developed to aid operating system development, in this case Unix. They were sort of developed side by side, and for what we were doing, building large simulation tools, it wasn't a real natural fit. But it is what it is, and it ended up working out for that company, and I don't think they ever moved to a higher-level language.

Hsu: So, you said you were there for two years. When did you move to your second job?

Naroff: So, when I start getting impatient with C and with the tools-- I had read some of Alan Kay's work on Smalltalk. There was a *Scientific American* article that was very famous that I found called "Microcomputers and the Personal-- " I forget exactly the title, but it was 1977, a Scientific American issue, and that was almost lifechanging. He talked about Smalltalk. He talked about Dynabook, which is

basically iPads back in the late '70s, and that really expanded my thinking. So, I started trying to talk up objects and better programming tools and languages to the principals at-- It was called HHB-Softron at the time. They changed their name. And they had basically told me, "Listen. We agree with you. Some of this stuff is interesting, but we're running a business and we don't have the time or wherewithal to retool everything. So, we'd love if you could stay, but we also understand if you want to go." So, I think at the time, job transitions were pretty much guided by headhunters. So, I believe I sought out a headhunter and said, "Here's what I'm interested in. Find me some companies that fit the bill." And I really don't remember whether it was the headhunter that found Cericor in Salt Lake City or I found them and then said, "Can you pursue them for me?" Don't really remember that detail, but bottom line is I found Cericor, Salt Lake City, Utah. And I just remember being really impressed with what they were doing because they, unlike HHB where you had a text file that would plug together all the components, they were doing it all on a screen with graphics. So, think of it as just as Apple helped everyone make the leap to a windowing system, they were doing this in the computer-aided design area, and their principals were taught at Evans and Sutherland and University of Utah, which was a hotbed of this type of work. So, even though Salt Lake City, Utah, was the furthest thing from a place I really wanted to be, growing up in New York-- I mean, it's really culture shock-- I decided to talk to my wife. We weren't married at the time, but we were planning on getting married, and I said, "How about we get married and then move out to Salt Lake City?" And that's what we did, and Cericor ended up being a great place. How much do you want to know about it?

Hsu: I want to ask more about Cericor, but let's-- Actually, you mentioned your wife. Could talk about--

Naroff: Yeah.

Hsu: Yeah, talk about meeting her.

Naroff: Yeah, so, I met Nancy-- Allen at the time-- at Oswego. Nancy was a violinist in the orchestra, and I was playing the Mozart "Clarinet Concerto," my greatest hit <laughs>, a solo with the orchestra, and through the orchestra and through-- I think we were in Music Theory class together. We had gotten to know each other and fell in love and decided-- She actually was a couple years ahead of me, so she graduated two years ahead of me, and then took a job at CBS Records and worked in the Legal Department, interfaced between the legal folks and many of the artists, classical artists.

Hsu: In New York City?

Naroff: In New York City-- Blackrock. So, that, too, was cool, because we got tickets to all kinds of great shows. We got to know-- in fact, just recently, Cho-Liang Lin, who's a very famous violinist, he was playing at a place my son was playing and we got to meet him, and he knew her from those days. So, it was like a reunion, and it was really exciting. And Harry Connick, too, is a big star now. Harry Connick was just-- He was a young gun at the time, and we got to know him. And so, it was-- We got to see Earth, Wind, and Fire in Madison Square Garden. Nancy would bring home stacks of albums for free, get all this music, so I'm like-- It was great. We were both really into it. Yeah. So, when I talked to Nancy about going to Salt Lake City, Utah, I remember her being very supportive. I think once we got there she had more

reservations, because at that point in time in that heavily Mormon culture, females were very much second-class citizens, and we were still young. And she had a hard time going from a great gig in New York to finding a great gig in Salt Lake. She did find a gig, but it wasn't great. And women at the time were not treated the way women were treated in New York. Let's just say that.

Hsu: Right. Right. Politically speaking-- So, it seems like you come from a fairly liberal background?

Naroff: Very liberal. Very working class. Very Democratic, overall liberal. Yeah. It's not-- I hate that there's such a division, especially today. There are certainly some good ideas from the other side, but in today's politics it's so tribal that-- Back in the day, though, I didn't pay any attention to politics. One of the unfortunate things, I guess, about getting older, is you focus more on that stuff, and it's pretty annoying in today's world. So, yeah. But in those days, we were liberal and believed in helping people, fundamentally, and believed-- We grew up in New York, and at the time, there were still a lot of people who didn't have enough. So, yeah, paying taxes and assuming that taxes are going to the right place and helping the needy, that was all part of it. And my dad grew up really poor. His dad was a cabdriver in New York, started the first dog-walking business <laughs> in New York. Unfortunately, he died young from smoking. But they were very poor. So, we have sympathy for people who have less, and when I look at everything we're talking about, I know that I'm standing on the shoulders of great people, and we'll talk about some of those later. But I also know that the time that I was born and the time I entered the computer science business was opportune.

Hsu: And earlier, you mentioned reading that *Scientific American* article from Alan Kay. How did you discover Alan Kay's work? Was it just by chance?

Naroff: I think it was just interacting with the great people-- At the time, I think it was HHB. I think we'd just talk about what was going on in the industry and people shared articles. Yeah, because no social media, certainly. No "Let's Google it." Like, here. It's easy. I found it online, really easily. So, it was just word of mouth. When *Byte* magazine was also very popular in the '70s and '80s, it was also-- could be-- Again, I don't remember exactly. Could be reading, let's say, the Smalltalk edition of *Byte*, and then reading names, and then searching out other things that those people had written or believed in. So, Alan was really well published. Actually, I didn't realize it-- Well, I did know it years ago, but now I was reminded when I was doing some research for this interview, that Alan graduated from the University of Utah. So, University of Utah is sort of an important place in computing.

Hsu: Yeah. So, did you actually ever visit there after you moved out there? Did you ever have any contact with the people at the--

Naroff: Well, I went out to interview, obviously, with the folks from Cericor and had a great interview. And it was kind of interesting; the person who I was reporting to was a Mormon, and he was very open about the culture, and I don't remember the exact words but was telling me that one of their attractions to me was not just some of the relevant experience or my passion for programming or the fact that I thought MAINSAIL sounded like a cool language to learn and work in, but he said something very unusual. He said, "One of the reasons we're interested in you is your leadership skills that are not only something we

believe you have, but sort of a cultural thing. It's like, you're from New York-- " There's something about being a Jew from New York that they were attracted to. There's this whole weird Zionist thing that goes on in the Mormon thing. So, I almost felt like there was some odd religious connection because of my heritage and where I came from. And listen, whatever.

<laughter>

Naroff: But he had said something to me like, "A lot of the Mormon folks that work here are worker bees and they're happy with that. So, realize that you're coming into a culture where you will be expected to lead and the people working for you want that," which was very different from NeXT, which we're going to talk about later. But I had no problem with that, and the group I led was great. High morale. Had a great time. It was probably the easiest management leadership gig I've ever had because it was so receptive to that, because a lot of Silicon Valley workplaces are like herding cats.

<laughter>

Naroff: Everyone's running around, has ideas, and wants to lead. But Salt Lake was different. So, yeah, we worked in MAINSAIL.

Hsu: That was your first management position?

Naroff: Formally, it was "leadership," but I think-- I wasn't-- I don't remember writing reviews, but I was giving people their work and doing a combination of design and just project management, and a lot of coding, too.

Hsu: So, you weren't formally a manager, but you were leading?

Naroff: They called you a "team lead." Yeah, I don't think I-- You know what's interesting? What's interesting about doing the little research I did for the interview is, if you would have asked me before I looked back in time how long I was there, I would have told you, "Probably two or three years." I was only there for a year and a half, and when I arrived only five to six months-- It wasn't quite six months. Five to six months after I arrived, Hewlett-Packard decided to buy the company, and that was also-- I would have guessed that was a year, but no, like really soon after the company was acquired, and it changed, obviously. And I had made a decision to stay under HP ownership for another year. But, yeah, MAINSAIL influenced me big time because they were so focused on trying to solve large programs. And they were funded, initially, by the Stanford Biology Department because the Stanford Biology Department knew they were going to have a lot of needs to write larger systems, and I don't think they wanted to do it in C. So, a lot of that funding ended up benefitting the CAE/CAD (Computer-Aided Engineering, Computer-Aided Design) industry because they were floundering, I think, with C. They needed something better. C++ wasn't out yet. But the features that were critical in MAINSAIL that made it so portable were-- Well, first of all, it wasn't just the language. It was a runtime, an environment, and it wasn't really object-oriented. It was more object-based. It was more module-oriented. I guess it was more like Simula and Modula and ALGOL. So, it wasn't-- A lot of object purists would look at it and say, "Oh, this isn't interesting from an

object-oriented perspective," and they might be right. But it was interesting for many other reasons, and it was one of these really pragmatic languages where they said you shouldn't have to link. There was no link step like C. Everything was dynamically loaded. You would load a module. As modules required other modules, they would be loaded. All the modules were self-describing. They all had metadata. That metadata was used for garbage collection and other-- So, it had garbage collection built-in, yet was very pragmatic with collection. It actually allowed you to do things that a lot of other languages didn't allow you to do, like get a direct memory pointer. Each module was tagged by the compiler, whether it was a safe module or an unsafe module. So, obviously in any large program you'd hope that most of your modules were safe, but it didn't stand in the way of doing low-level bit twiddling if you had to. So, it was a very pragmatic language, had a coroutine facility, which [is] similar to blocks in other languages, and they did great things with it.

Hsu: MAINSAIL came from the Stanford AI Lab?

Naroff: So, SAIL was developed at the AI Lab, and then the Biology-- I don't know exactly how the connection was made, but the Biology Department at Stanford then hired the principals at Xidak, which was the company that was formed out of Stanford to work on a machine-independent version of SAIL. See, people were struggling with all these computers that were coming out. There was so much diversity with workstations. Sun, for a while-- well, Apollo before Sun, which was a Boston-based company, they were pretty big for a while. They did the AEGIS operating system. But then when Unix hit, even AEGIS had to offer Unix emulation. So, yeah, with all these workstations out there, companies just wanted to be abstracted from this, and so they write a program that runs everywhere. So, yeah, MAINSAIL was financed by the Biology Department, and then later they spun the company off, and I don't know how that happened. I don't know any of that history, but it's interesting. They were located on Deer Creek in Palo Alto, right almost beside-- where NeXT was [later]. I thought that was a really weird coincidence, right?

Hsu: So, MAINSAIL was a portable version of SAIL, essentially?

Naroff: Yes. That was a stated goal, to develop large portable programs, and it did that really well even though it never gained wide acceptance.

Hsu: But it was taken up by-- you mentioned CAD companies like Cericor-- because it was good for-- Why was Cericor-- Why did Cericor decide to use MAINSAIL, maybe, is the question?

Naroff: I think the choice was between C and, in their mind, MAINSAIL. There wasn't any other choice for them. They had found MAINSAIL and liked the idea so much after looking at the feature set, because realize, most garbage-collected languages-- let's say Smalltalk-- were born in a research environment, and they weren't pragmatic for real products. So, I think that they thought they could influence it. They thought they'd get special attention. They weren't going to be waiting around for C++ to mature, which at the time C++ was being talked about and being researched, but there were no implementations which people felt comfortable with in the early '80s. It was a pre-processor still, not native compiler, and the language was just growing and evolving, and I think they felt like they could work with Xidak. Xidak did give Cericor a lot of personal love and care, which was important because they were a real company.

They wanted to develop real products. And yeah, I think they were happy with their decision, but HP wasn't happy with the decision. So, HP came in and started pretty quickly getting fearful of dealing with such a proprietary language with such a proprietary company. And obviously, they could've potentially bought it and fed it, but they decided against that and they started this five-year-long effort to convert the MAINSAIL code to C++. And I knew from the very beginning that this was not looking like a smart thing for them to do, and five years later they canned it. So, they tinkered around for five years-- I wasn't around at the time-- and eventually canned it. And it's one of those naïve management engineering decisions that you see far too often. So much wasted time and effort because executives make decisions that make no sense. And they weren't asking for our opinion.

<laughter>

Naroff: I'm sure I gave it to them.

<laughter>

Naroff: But they didn't care. Taking an application which was developed with garbage collection from the get-go and going to a system that isn't garbage collected, just that alone is a showstopper, but then if you consider other showstoppers, like no-link, dynamically loaded on the fly, C++ wasn't very good at that, and yeah, it was just such a bonehead decision. Man. And I don't know when Xidak went out of business, the company that did MAINSAIL. They were just great compiler people. They were another example where, they're a great technical startup that other great technical startups admired, I think, because when I talked to Mentor Graphics who was big in that area and even interviewed with them later, they had tremendous respect for MAINSAIL and what we had done at Cericor. They weren't as brave, and their technical people acknowledged that, and they were moving to C++. But they were moving to C++ from C, so it was a much easier transition for them. But Xidak failed like so many startups, especially language startups, where they might have great compiler chops, great insight into runtimes and compilers and linkers, and they did, but the business plan wasn't as well articulated. And let's face it, being in the language business is a tough business. I watched your Brad Cox interview, and Brad was very clear that it was tough being in the tools business in general, but the language business in particular. Yeah.

Hsu: One last question, I guess, before we move on from Cericor. Cericor, that's C-E-R-I-C-O-R, correct?

Naroff: Correct.

Hsu: Could you talk more about the product that you are working on for Cericor?

Naroff: Sure. Well, it was a schematic capture package where people would design circuits. They would plop their little NAND gate or whatever their gate is on the screen, and they would wire it up to other components and then could package those components, and so on and so forth. So, it was a graphical development environment for designing circuits and simulating them. And it's so cool. We would do circuit simulation, and you would see the zeros and ones flow through the little wires that were on the screen. Very, very sexy, very cool product. Unclear if it would scale up to the type of demands that really, really

large circuit purveyors have. That's also true for startups. There were a lot of cool features. I don't know whether it was robust enough to support the type of things Intel does. Clearly, Intel and other hardware companies have a lot of homegrown tools, some of which I imagine are off the shelf. But ultimately, it's a combination of things. So, that was, at the glass, what the product was, but what I was working on was-- One of the projects I did involved capturing all the input so that we can do testing of an interactive tool like that, because with interactive tools it's usually harder to write regression tests. In other words, you want someone to build a circuit and then succeed. And then, as you evolve the system, rerun that schematic capture and simulation. So, one of the projects I had was to capture all the menu input and mouse input and play it back later. So, we called that journaling, at the time. So, journaling was something I worked on. I also worked on the object-oriented architecture that was built on top of MAINSAIL. As I said, MAINSAIL in and of itself wasn't wildly object oriented. So, they had a system which was designed before I got there, but I also obviously contributed to it when I got there, to develop an object-based architecture that was used to develop the product. So, for example, we didn't have an Interface Builder to build the GUI itself, but we did have a language that allowed you to express the UI so that you wouldn't have to write it in barebones MAINSAIL code. I think we called that the Cericor Development Language. I don't believe we published it to our clients, but it was used internally. So, that's where I got a big taste of actually metadata, designing an architecture on top of a language, and my job was to learn it, then educate all the team members and evolve it so that it had integrity because there was a lack of knowledge throughout the company, especially when HP came on board. When you start adding more and more engineers about, "Well, how do I do this? How do I add this or do that?" So, yeah, I led the infrastructure team. I forget what we called ourselves. And there, I was a leader, and they wanted to bring me into management, but as I said, I only stayed a year and a half at that company, and it's kind of-- I look back at some of this stuff and it's just amazing how much was packed into some of this time <laughs>. A lot of stuff. Things were moving fast in the late '70s, early '80s, late '80s.

Hsu: So, clearly you saw the writing on the wall, at some point into the HP acquisition of Cericor.

Naroff: I saw the writing on the wall. In fact, after the acquisition, I took an interview at Mentor Graphics in Oregon; that's where they were based. Mentor was the 2000-pound gorilla in that space. Very successful company. I think they're still around. They were sold for billions of dollars not so long ago. And they made me an offer. And I decided, rather than leave Cericor after merely six months and go to Mentor, to be a little bit more patient. And at that time, I actually made the decision that I wanted to get out of the software tools for hardware people and into software tools for software people.

<laughter>

Naroff: Because I was a software person. I pretty much made the decision that "Mentor's in good hands. They don't need me." They thought they did. They didn't, and I said, "Let me continue helping HP even though I know it's not heading in the right direction. There's still stuff to do, and there's fun stuff to learn. I like the people." So, I hung out another year, and that year was the birth of OOPSLA. So, I went to OOPSLA, the very first one, back to Portland where it was located, and that was where my idea-- That's where "Opportunity is where luck meets preparation-- " I was prepared to make that transition, and I met a lot of cool people, including Brad Cox and Tom Love, at that first OOPSLA. And I don't know whether we

decided after that brief meeting at OOPSLA to have me in for an interview. I don't remember exactly how it unfolded, but soon after meeting them I took an interview in Sandy Hook, Connecticut, for them. It's interesting to note, before making that connection to Objective-C and to, at the time PPI-- Productivity Products International-- I thought I wanted to work at AT&T on C++ because they were doing really cool stuff. And AT&T, I think Bjarne [Stroustrup]'s lab was a lot of their research work in Princeton, New Jersey, or near it, and it's a beautiful place. It's a great place, and it's near where I grew up, and so it's like, "Wow. I could go back to where I grew up." But long story short is, the little bit I pursued them, they basically immediately-- I forget whether it was a headhunter or they said directly, but I didn't have a Ph.D. So, "We don't need you." <laughs>

Hsu: Because it was Bell Labs, right?

Naroff: It was Bell Labs. It was Bell Labs. Only Ph.Ds need apply, which at the time I was actually pretty upset. I wasn't so upset I wanted to go back and get a Ph.D.

<laughter>

Naroff: But it was a ding, considering everything up until then was, "Oh, yeah. Come on. Come work for us," and "Wow, you're doing good work." That was my first kind of rejection, but as with lots of things, things still worked out. Because if they would have accepted me, life would have been a lot different, I'm convinced. So, fortunately, they rejected me, and I went to PPI, had a pretty good interview, and they made me an offer. And Connecticut wasn't too far from where I grew up. So, it was still-- I was able to see my parents and feel like I was back on the East Coast, because Salt Lake, again, was culture shock. So, that was good, and--

Hsu: Was Nancy able to find a better gig? <laughs>

Naroff: She did. She did. She did. Actually, when we were-- Yeah. I forget what the company was called. Or maybe she worked for a physics professor. There was some physics lab there. I'm drawing a little bit of a blank on where she worked. But we liked it. We also bought our first house, a condominium. Interest rates were 14 percent.

Hsu: Woah.

<laughter>

Naroff: Imagine that. Imagine that! Fourteen percent. Yeah. I guess it encourages the houses to go down in value. That's the good news. You pay less for the house, hopefully. But yeah, we moved to Connecticut, in Sandy Hook, and the location of PPI was so beautiful. I wish I had more pictures in the day. So, that was my transition into working on software tools for software people.

Hsu: So, yeah. You said you met Brad and Tom at OOPSLA. What was your first impression of them?

Naroff: My first impression of Brad was how composed a speaker he was. He was really composed on panels. I think I saw him on a panel. And very professional, well spoken, and at the time, most of what he was speaking about was not Objective-C, the language, but the revolution that he saw with evolving to object-oriented. And as you know he wrote a famous book, "Object-Oriented: An Evolutionary Approach," and I had picked that up, and I wasn't as smitten with the book. I thought it was a little bit more marketing than it was technical, but that's not necessarily bad. It's clear that he had a knack for selling the vision, and he talked about the Industrial Revolution, and he talked about soldering ICs together. And there was something compelling about what he was saying at the time, and I don't think I was the only one who felt that. I think his schtick put PPI on the map. Tom was more in the background. Tom was more the business guy. Tom was also a very charismatic speaker, but he was typically speaking to business people, not technical people. But Tom and I actually lived in the same condo community and got to know each other quite well, and he's just a fantastic human, and I really like Tom. So, I liked both of them when I met them and felt like they seemed-- though they were high-minded and aggressive, they seemed to have some pragmatics as well, which I really wanted to work for a company that wasn't too ivory tower, but wanted to work for a company that was lean and mean and trying to do stuff for real people and real projects. I wasn't interested in research. Research was something that I had avoided until that point, and I just felt my personality type, which tended to be a little impatient and a little emotional, was probably best in a product environment. Or, I should say, maybe even just better recognized in a product environment. So, I felt it was just a better fit. I ended up being a little wrong with Productivity Products [International]. Meeting them and then going to work for them ended up being different, so I don't know if I got an accurate picture. I don't know whether I was green and just so smitten with the idea, because the whole idea of making or evolving the C community to an object-oriented programming language was so desirable to me, because I had seen MAINSAIL take a more radical approach, and I saw it fail. So, the idea of taking a radical approach-- Because I was really disappointed with what had happened at HP and MAINSAIL and Cericor. I mean, let’s face it, moving cross country, I put a lot of my soul into that, and it really was a big disappointment. So I wanted the next opportunity to be less disappointing. <laughs>

Hsu: So part of it was the vision?

Naroff: The vision and the pragmatics.

Hsu: Together?

Naroff: Together, yeah, and I liked both of them.

Hsu: So you joined PPI, it was still called PPI then, in ‘85?

Naroff: Yeah, I forget when the name was changed to Stepstone but it was coincident with the venture capitalists coming in. I forget when the venture capitalists and Tom Love parted ways, but they did part ways. That was tough for me because Tom was a mentor of mine.

Hsu: So he was forced out? By the VCs?

Naroff: That's my recollection. But I think Tom is better prepared to ever talk about the details. I spoke to Tom fairly recently, and I don't think he felt he was forced out. But it almost doesn't matter. The perception of the company was that the venture capitalists came in and forced Tom out. So that's my perception of it, and, listen, it's just like some people perceive that Steve was ousted from Apple and some people perceived he wasn't. Some people perceived he was just put into a place where he would then want to leave, which could've been the same situation with Tom, for all I know. It could've been, "Okay, well, you can stay and do this. And we know you won't be happy so you'll leave." So the result is the same.

Hsu: Interesting. Could you maybe describe how the company was before, when Tom was still there, and how big it was? What was the culture like? And then maybe talk about how things changed after the VCs came in?

Naroff: To be honest, Tom was not around PPI a lot at the time, at least that's my recollection. He was always out trying to do deals. He was prolific in the number of companies he made contact with, which is one of the reasons it was bad that he left. Because he was really great. HP Labs, for instance, was a customer. I'm pretty sure Tom made that happen. So I don't think Tom had much influence on the culture of the company because he was always trying to do deals. He was very outbound, which, to be honest, was disappointing to me, because here's a guy I admired. I almost wanted more interaction with him. So the other person I had interaction with was Brad, and he was closer to what I was working on so it made sense that he was around. But he was focused on other things. He pretty much handed the language off to me and decided that he was going to go work on other stuff. So Brad and I ended up not having that much collaboration, because he trusted me with what I was doing, and he was interested in other stuff, so Brad and I, too. So there was a void in the company with leadership/management, because the guy that all of Software reported to, and I think all of Software, just to quantify it, was probably 12 developers. It's fairly small, maybe less but my recollection is somewhere between 8 and 12. Sometimes we'd hire consultants temporarily to help out. So we had a manager of Software who was actually the person who signed my offer letter, who went away soon after. Then they hired someone from ITT that they felt comfortable with, because a lot of the folks who founded PPI were ex-ITT folks. So they hired a fellow named Ken Hamer-Hodges who was running Software. Ken, I think, had a technical background but wasn't really driving Software. So Brad was off in his ivory tower, Ken was the manager, Tom was off trying to get stuff done. You had a guy like me trying to lead the future of the language. Then you had someone working on the interpreter and a few people working on ICPaks. It was pretty scattered. It was not well managed. It was probably the least-managed company of the three companies I had worked with to date. The first two were extremely well managed.

Hsu: So you mentioned Brad was doing other stuff. Was he working on the ICPaks, the libraries?

Naroff: He was working on ICPaks. He was also, I believe, either before or after, working on blocks. Now he was doing blocks as a library component without language support. That work wasn't really in any way being coordinated with Development. At the time, at least my recollection was, Brad's door was usually shut and not really open. It was really distinct. It's almost like he was the research wing, him alone, and everyone else was Development. At some level, that's not necessarily bad but there was a

void. Because, often, you look to the visionary founder for at least some direction. But there were enough problems with the prototype implementation, the first and second iterations of Objective-C that there was plenty of stuff for me to work on and identify. So my decision was to just put your head down. My first six months were spent porting it all over the place. There was no documentation on it whatsoever so I documented the architecture and the implementation so that we could make decisions moving forward. Part of that was where I figured out, "Oh, my God, some of this stuff is really flawed." Porting, at some level, I felt like, oh, my God, I was doing so much interesting work at Cericor with architecture and objects. Here I am porting this preprocessor to all kinds of weird architectures, and finding all kinds of problems. It did feel a little bit like sweeping the floors, like I wasn't moving in a totally upward direction. But it didn't really matter to me because I believed that there was so much opportunity.

Hsu: Well, let's get into the state of the language, when you started on it. So you've mentioned to me before that you considered it a prototype. So what was the state of Objective-C and the compiler, if it even was a compiler or not when you came in?

Naroff: Yeah, Brad talks a little bit about this. Calling it a compiler was probably loose. It was more like some scripts that would key off certain things and then generate code.

Hsu: Generate C code or generate machine code?

Naroff: It wouldn't even touch the C code. It would just try and touch the Objective-C parts and translate—expand them in line and leave the C code alone.

Hsu: So it would just translate the Objective-C into C?

Naroff: Exactly, along with the other C so that a C compiler could accept it.

Hsu: So then you would feed it to a regular C compiler?

Naroff: Precisely, yeah.

Hsu: So it was just a translator?

Naroff: It was a translator. Now, translator's a very general term. Compiler people use that for lots of different things. But it was naïve. It wasn't full-bodied. It was not capable of detecting programming errors that are common, even a misspelling. Part of the reason it wasn't detecting errors was not only the technology but the language definition. There was no explicit interface declaration whatsoever. So if you typed a message expression and sent an object a message of a certain name and typed it incorrectly it would just assume that that was the name of the method. Yeah!

Hsu: Ouch.

Naroff: Ouch! Well, especially when you consider the compile on debug-link-debug, that whole chain was slow in those days. So, yeah, it wasn't pretty. So, after porting, two things were clear. We needed to evolve the language as minimally as possible, from my perspective at the time, for it to be reasonable. The goal wasn't how do we make the greatest language. It's like, okay, we're in a bad place right now just from some very basic places and that we need to add stuff that minimally enables the compiler to detect errors. And we need to write a compiler that's capable of processing the language in its entirety so that it's far less naïve. And, lastly, we need to make sure the code that's generated is robust, because they were trying to work with Make and other Unix tools. So just like Objective-C was a compromise from the beginning by saying, "We're embracing C." I mean, as soon as you say you're embracing C, you're compromising, from a purity standpoint. With that in mind, they were also impure in the sense that they wanted to work with Unix tools, like Make. They didn't want to have to build their own build system and so on. So you had to be sensitive to, if the compiler's going to be generating a file and make the text that file, but doesn't recompile the right things, and you produce an executable that doesn't work, that's really bad, and the compiler was doing that. The compiler was actually, along with the Make system, often you'd link a program and it would crash, and you'd find out, "Oh, wow, it's because our intermediate file isn't being managed properly so that Make isn't recognizing that that file needs to be regenerated," and bad things would happen. So to me it was a mess. So I wrote a proposal with a minimal set of things, along with an engineer named Alan Watt, who was, I believe, also an ITT engineer. We wrote this paper and part of the paper was, "What are some of our other customers asking for?" Like HP had developed a garbage collector and they wanted us to integrate their garbage collector. So on one end of the spectrum you had researchers who were asking for things. On the other end of the spectrum you had a company like NeXT who was developing an operating system and a set of tools. They were more sensitive to the type of problems I was just talking about. I fought pretty hard for fixing the basics, which, to me, it was as obvious as, "Oh, my God, recrafting this MAINSAIL program in C++ is heading in the wrong direction. It was as obvious as that, yet I got pushback from the venture capitalists who were thinking in dollar signs, thinking, "Why should we pay you to recraft and add some of these things that will make it more robust?" To them, error-handling—they're money people. This notion of, "Well, we can't flag an error," to them, that's like, "Well, so what?" <laughs> They're not programmers. So I was getting a little bit frustrated when I was basically being somewhat ignored. There was some appreciation but this person, Ken, who was leading Software, he didn't have the technical chops to, as the guy who was running Software, to fight for me. I had to fight myself. So the great fortune was NeXT arranged a trip for two of their engineers, Steve Stone, who was an OS guy, and Trey Matteson, who was just out of Brown, really bright young guy. They both came to visit us in Sandy Hook and read my proposal and were just jazzed, totally jazzed. They're, like, "Oh, my God. This is exactly what we're bumping into. This solves our problems." And Steve Stone was a bass player and he and I hit it off. There are lots of obscure groups that I was listening to at the time that Steve was totally plugged into. We went out to Greenwich Village, had a great time. So it ended up being a really easy meeting because I had it all laid out there for them. But I said, "Guys, when you go back to Palo Alto I need you to write a letter that identifies what we talked about and why you were supportive of this, so I can basically do the political dance <laughs> at Stepstone." And they wrote the letter and I have the letter to this day. I think you've seen it. After that they had to let me work on this stuff. So that was how I got to add interfaces and fix some of the runtime problems that we've talked about before. I mean, I just described some of them. An object module not being recompiled but creating a bug because of the poor language and compiler tools, that's clearly bad. In

another interview, I've talked in gory detail about that but it's probably not necessary for this context, right?

Hsu: Yeah, probably, maybe just in a sentence or two.

Naroff: There used to be a global pool of selectors, which are the names that's used by Objective-C. That global pool was contributed to by the object modules. The linker would, basically, coalesce things. If the table that was in the object modules was not up to date then when it got copied to the global pool the global pool would be corrupt and the program wouldn't run. Simply put, it was a good prototype. It was a toy in the early days. My job was to try and build it into a tool that had more wins than drawbacks. Yeah, the letter worked. I got the freedom to recraft the compiler from the ground up and solve all these problems. Even on the machines of the day, which were typically running four megabytes, the performance of it was not that bad. The performance of launch time and the more dynamic runtime I added was work that we did throughout many years. At the time, some of the resistance I got, not only from the venture capital people but the engineers, was, "Can we really do dynamic relocation of some of these big data tables?" And I said, "It's a dynamic language. It has to be implemented dynamically." You don't implement a dynamic language statically, you just don't. Sure, there are some static elements. I'm not saying all class descriptors need to be cons'd up on the fly, you can imagine that. The class descriptors were laid out by the compiler and coalesced by the linker. Even that could be dynamic. You could read a text file with all the attributes in it and actually generate the data structures on the fly. So it was a really delicate dance between what does the compiler and linker do and what does the runtime do? But because I had worked in a rich runtime like MAINSAIL and they were always trying to understand that delicate balance, I had some relevant experience, not as a compiler writer but as a language user, which also includes metadata. The original metadata that was in Objective-C was incomplete, so I had proposed making sure the metadata, which is the data that describes the classes and the instances and the methods and so on, that that be much more robust. That also was something that evolved over time.

Hsu: Earlier you mentioned porting the language to various places. What sorts of platforms were you porting to?

Naroff: Oh, man. Well, HP-UX, Apollo, the most obscure machine I ported to was a Gould, which was like an IBM clone minicomputer. It was really weird because I had to generate, I don't remember the exact details but it was something like the assembly language was accepting VAX mnemonics but they were then being translated to IBM instructions. I was, like, "Wow, that is so weird." It's because whoever wrote the assembler at the time took the code from somewhere else and just hacked it and stuff like that. But, yeah, the Gould port was really weird. There wasn't really much assembly. That was another requirement when I would dive into this is there's various ways to implement message dispatch. The way they originally implemented it required some assembly language. Because you call a function, let's say message send, and you're in the message send routine but that message send has arguments that need to be passed to the eventual object. You have to leave those on the stack frame. So you need a way to jump to that, which is really only done in assembly language, at least when you put the message send arguments and the user arguments on the same stack frame, which is what we were doing for efficiency. Another way to implement messaging is to have the message dispatcher or the message lookup routine

return a pointer to the target and then call it again. So you have two function calls. So it's a classic difference of speed versus elegance. One was easy to port, slower, two function calls, one is one function call, requires some assembly hack, but it's faster. And, obviously, when you don't care about portability and you're developing an operating system like we were at NeXT and Apple, you just do whatever's the absolute fastest on your processor and operating system. But Stepstone was in the-- we don't own an operating system or processor. So there were various techniques. And, often, what I would promote is, "Let's get the port up and running with the slow mechanism and if we sell enough of these we can go and do the assembly work to make it faster."

Hsu: Right, yeah. Makes sense. You mentioned NeXT and HP, what were some of the other customers that you were doing these ports for, and how did you get these customers?

Naroff: I think Tom, at the time, originally. And how did we get the other customers? I mean, they were advertising in Byte. I actually found one of the old ads and copied it for you. They advertised in *Byte*. *Byte* was really well read. So I think a lot of people contacted them, and hopefully we had a port they had. If not, they'd offered-- That's the other problem with a small company. What if someone calls you and says, "I need one for this processor," and it's one sale. Well, in a desperate company, you allocate resources to develop that port for that one sale. In a non-desperate company, you think, "Well, let's at least have some critical mass of clients for that machine." <laughs> For example, I can't imagine they sold that many Gould translators. <laughs> All that business stuff, Hansen, was out of my hands. All I know is they asked for some of these ports. I don't really know how they got the customers. I actually never interacted with that many customers. We were pretty sheltered. Even the NeXT interaction, I felt like that was fostered by me, given my position that I've described. They did not really want developers to talk much to customers, or to see what was going on there, which I don't think is that unusual, necessarily, yeah.

Hsu: How did both HP and NeXT become customers? Was that through Tom as well? How did they first become customers?

Naroff: HP, I don't know. I wouldn't be surprised if it's as simple as-- I know it was HP Labs, so it was not on the mainstream of HP's product cycle. So I think it's pretty common for researchers to buy tools, try them out. If you're a researcher and you've inherited a big code, C base of code, "Hey, look, I just saw this advertisement for Objective-C. It gives me some Smalltalk with C." So I think a lot of it was pretty casual and organic. I don't think anyone went out and-- It became a bigger deal when we were trying to sell the ICPaks. Because then you're selling an architecture, a set of components, and issues like source code, do people get source code, that's a bigger issue. People usually don't like being in the language business. But when they start using actual software they become a little bit more antsy about making sure they get source, and then it's a bigger rat hole, which Richard Stallman has the answer to. But not everyone was willing to give away their source code. But let me make sure I answered the question.

Hsu: How did NeXT first become a customer?

Naroff: Oh, so I wasn't there, obviously. I was at Stepstone, but I'm told that William Parkhurst, who was the original AppKit developer, was smitten with it. I guess he, maybe like I went to OOPSLA, I don't know that William was at OOPSLA but I think he may have been. And he was looking for tools and came across Objective-C and figured they'd give it a shot. I think they were also considering Object Pascal or Pascal. I'm not sure Object Pascal existed but I do believe they were considering some of that. I've heard from some NeXT folk that there was some discussion about using display PostScript directly because Bud Tribble thought the overhead of Objective-C might be too great. Bud was running [NeXT] Software at the time and ran [had run] the Mac project [at Apple]. He was one of the very early NeXT managers, leaders. But they decided to do quite a bit with Objective-C and hit these walls, and also hit compile time walls. There were lots of issues after they had spent, I think, six to nine months with the language, roughly. That's when they basically contacted Stepstone to read them the riot act and say, "We've made a big decision to go with your language and we need it to be pushed in this direction or else we might reconsider." It really was born out of good politics. The politics were valid, it wasn't arbitrary. It was bottom-up. It wasn't like Bud was driving this. The engineers were driving it and probably really making William unhappy. The engineers at NeXT were, like, "William, what did you do to us?" I don't know that, William and I correspond occasionally, guess I could've asked him, or should've asked him. But I could just imagine, based on when I ended up doing some consulting for them as a Stepstone employee and later hiring on, I knew they were in pain. They were in pain. But, let's face it, C was at the core of Mach and their operating system decision. The only path for them was to just make this work, and that's something that I sensed big time after talking with them. It was so clear that this was a major obstacle that needed to be solved. And it's funny, the venture capital people, I don't know whether it's an East Coast, West Coast thing, but their respect for Steve and what NeXT was doing, it's almost like, "Well, why is NeXT more important than HP Labs?" To me it was so obvious. How could you even question that? HP Labs is piddling around with the language and NeXT is building an operating system. This is our chance. It didn't seem like they got it. Again, it could be East Coast, West Coast because, to them, "Oh, well, yeah, sure, Steve Jobs founded Apple but, yeah, that was what it was. So he's starting a company, so what?" To them, it felt like, "So what?" I was not-- I didn't grow up with Apple. I was not a Mac person. I was weaned on, first, as we talked, mainframes, then minis, then workstations. The reason NeXT excited me was, wow, a workstation that mere mortals could use was really exciting. But PCs and Macs, to me, were toys for what I was working on.

Hsu: So the VCs saw HP Labs, that's a big name.

Naroff: Big name, exactly.

Hsu: And this NeXT startup, who knows whether that's going anywhere? They didn't think that was going to be a big deal?

Naroff: Right. Yeah, there's a great video that I'm probably going to mention later, if we get a chance, by Malcolm Gladwell. It's a 20-minute Google presentation where he talks about elite institution cognitive disorder. Have you ever seen that video?

Hsu: No.

Naroff: Oh, my God. It's brilliant. I think Stepstone had elite institution cognitive disorder. I think, from their perspective, HP was an elite institution, and it was. HP, back then, they were so respected. Things have changed. But, yeah, to them, NeXT was just this startup company, just like us. We're a startup, they're a startup. <laughs> Actually, the Malcolm Gladwell talk, what he's talking about there is how people get so smitten with elite academic institutions and often will mortgage the house and everything else to go to one of these big-name schools. They go to the big-name school, especially in STEM, and they fail. They fail, not because they're not smart people, because if you get into Yale or Harvard or any of these really big schools you're obviously a smart person. But they fail because they end up in STEM or computer science or name your tough undergrad degree, they end up being in the bottom third of the class and that's never happened to them before. They're convinced they're a failure. He backs it up with data and he tells an interesting story. Because he, too, didn't go to an elite institution. He even makes fun that one of the reasons he agreed to give this Google talk, which was for free, usually he gets paid, was that it's for Google. He's really smitten with Google. He even said it's not about even hanging out with really smart people. "I know plenty of smart people." <laughs> It's a great talk.

Hsu: I'll check it out. Before we move onto NeXT, let's finish off the Stepstone story. So you mentioned the VCs came in and then the company changed. Could you talk a little bit more about that?

Naroff: Well, it's a little bit hard for me to quantify since the culture at Stepstone was already not wildly good. I sort of described that before. In other words, we lacked real leadership. Brad was off doing some research. The management was from IT&T. It wasn't a hungry entrepreneurial startup like the other two I had been a part of. When you take that lack of focus and lack of hunger and you bring in money people who think they know better than the engineers, it's not a good feeling to be part of an organization like that. Because engineers, along with the customers of the technology, should be driving the technology. They need to be sensitive to deadlines. They need to be sensitive to, "Well, if we do that we only have one customer for it." There's that type of management, of priorities, that wasn't happening. We had a small staff and we were doing too much, broadly. So the venture capitalists, because they want to make more money, they view more as better. So let's do more ICPaks. They didn't understand that even the ICPaks we were doing were half-baked. They didn't understand, and something I learned, I didn't know this at the time but now, looking back, I learned it soon after Stepstone, actually at NeXT, and I've never gone back, is you cannot develop a domain-specific ICPak or kit, whatever you want to call it, or ObjectWare. There's been many of these terms for grandiose libraries that solve domain problems. You cannot do one of those without doing an app to prove it. I would argue, two apps, at least. Because if you do a kit and do an app, well, that says, "Okay, that kit's good enough to service that app." Let's say it's a spreadsheet. But then if you say now you want to do a schematic capture package, wildly different. Would that graphics model support that application as well? They didn't understand this. And I was pretty green at the time. Again, I'm not saying I understood it either. But if they were well-managed and they had the right people at the helm of the company, someone should've understood it. Brad didn't understand it. I mean, I haven't talked to Brad recently but I saw the interview you did with Brad a couple years ago. Brad just thinks ICPaks were ahead of their time, or you can't sell them or it's destined to failure. Almost like, "Well, we tried and there wasn't a business model." Well, you can blame the business model all you want. The technology just wasn't good enough. And there are business model issues. I'm not suggesting there aren't. There's lots of issues. But fundamentally, if you're talking about a

company like NeXT who's doing professional stuff or Adobe that's doing professional stuff, or Lotus, the guys who are doing pro apps, they're not going to base their future on your objects where you have no apps, especially when you're not giving away the source code. So, they didn't understand this. The venture capitalists, they just wanted more ICPaks. They really didn't want to support the whole idea of working on Interactive Development Environments, better compilers, browsers, because let's face it: one of the great things about Smalltalk was it had a great browsing environment. And that's something I worked on at NeXT later on, but when I was talking about that at Stepstone, "Well, we're not going to do that." So, their focus was almost entirely on ICPaks, which is also what Brad's book was about. The revolution. "Hey, we're going to develop-- " And it's actually interesting. There's a talk I found a few weeks ago called "Object-Oriented Programming is Bad." When I first saw it, I was like, "Oh, my God. This must be really weird," but it had a half-million views, and so I said, "I have to watch this if it has a half-million views," and the guy who did it is really pretty thoughtful and it's an interesting piece. I don't agree with all of it. He absolutely is down on encapsulation. Absolutely thinks that it will lead you to the wrong place. I don't agree with that. I think you can use it effectively. But one of the things he talks about is how he always felt, when programming with objects, that it's like designing a house with lots of walls before you really know what you're using the house for or the building for. That struck a chord with me, and I think to some degree it relates to this discussion where Stepstone was making these plans and architectures, and other companies were guilty of it, too-- Taligent in the future-- where they're building all this stuff, but they really don't know what's going to be sitting above them, and it's almost too soon to hardwire the architecture.

Hsu: So, they're building tools, but they haven't talked to who's using the tools, so they're building it without an idea in mind for how they should be building these components?

Naroff: That's right. Even something as simple as a hash table or a lookup routine-- You can build, obviously, a hash table object without writing an app. However, the different apps will or will not work well with that hash function or hash table. So, by using that hash table in many different scenarios, you will parameterize it differently, you will build it differently. Even the hash table for message sending, which we may talk about a little bit later-- I went with the simplest thing on the planet: linear probing with a particular hash code that was computed from the selectors. And we-- Bertrand and I and others in the early days [of NeXT]-- we played around with lots of different schemes, and the simplest thing is what worked the best for Objective-C. It's not intuitive, some of this stuff. You have to build lots of different things and measure them, and there's a lot of work involved in that type of craftsmanship, which clearly wasn't happening at Stepstone. It was purely, "Listen, we need this. We need that." And unlike the-- and it's interesting, like the open source movement, one other thing that's wonderful about it is people will get it and there'll be more of this loop and collaboration. That's the other thing that wasn't happening that much. Like when HP contacted Stepstone to politic for taking their [garbage] collector, they literally wanted to just give us the collector without any iteration with other customers. There's another perfect example. If we integrated their collector, yes, it might have made them happy but it, I can guarantee you, would not have worked for NeXT. And so, I felt like-- just to tie this up-- I felt like the first two startups I worked for were professional companies with professional people, and I was treated professionally. And at Stepstone, though I really love Tom and like Brad, the company was dysfunctional, and the fact that they didn't make it is not surprising to me at all. It's not about the concepts. They had the right concepts. They didn't execute the

business, they didn't execute on the technology, and that was their problem. And that's why it was interesting, when I was listening to Brad's interview with you, I was a little bit animated, and like, "No, Brad. There's other things." But Brad's a brilliant guy, and Brad just-- There were so many business things you asked him that he really couldn't answer. So, he was divorced from that as well. But in a startup company with very few people, the principals need to have strong opinions or else other people will come in and ravage, and that's what happened.

Hsu: So, before the break, I guess I just wanted to ask one last question about-- this is not Stepstone particular, but we mentioned the idea of ObjectWare, which I guess came from Brad Cox's vision of software components as being a revolutionizing force in the software industry. And I think your take on that was that it was not necessarily the vision itself that was the issue, it was the execution, but I was curious about your take on the whole idea of selling objects on the open market and the potential impact that that might have. Since that was clearly Brad's vision, it was-- Maybe part of that vision was what motivated you to join Stepstone in the first place. I don't know if that's true or not, but maybe just talk about what your view of that vision was.

Naroff: At an abstract level, it was definitely part of it. I, at the time, wasn't a real deep thinker in that area, but as far as the sales pitch goes, I was partly sold and just skeptical. And I think I didn't really put any meat on the bones while I was at Stepstone because I had my hands full with trying to save the compiler and language. If the compiler and language were in a better spot, I probably would have been more intrigued with Stepstone's decision making in the ICPak area, but it was almost like it was going on as a separate thread <laughs> within the company that I just didn't pay much attention to. And in fact, not only did I have my hands full, but the culture of the company encouraged people on different threads not to communicate, which wasn't the culture at other companies, but it really was the culture there. In other words, I had some ideas on the interpreter, and I was really never encouraged to participate in interpreter plans, which clearly the language changes I was proposing did affect the interpreter because it was yet another implementation of the language. It was just more interactive. But the interpreter was also a half-baked-- When you hear Brad talk about the company, he doesn't really talk much about the interpreter, and if I recall, a lot of that may have even been inherited from another company <laughs> that they were working with. So, it was just a hodgepodge of stuff. And as we previously talked, looking back, I know very clearly-- And I don't need to be pedantic about it unless you want me to clarify something, but I just think it was the execution. I also think-- I think I said this before. With components, shipping software becomes pretty important because ultimately if there are nontrivial components, companies don't want to put their future in another company's hands without control of the source code, which is true for other stuff, too, but it's particularly true for components.

Hsu: Right. And in some ways, it seems almost as if Brad's model of selling objects-- I guess you could modify it to include the source, but his model was that it would be completely black boxed and you wouldn't get the source, so you were stuck if there were problems with it.

Naroff: Exactly, and if they-- If Stepstone was more active in app development, they would have had more combat experience with those realities. So, I think they were just naïve. I think it was naïve, and I don't know why it was naïve but sometimes people just have to experience things and move on. There

was just never any thoughtful-- I don't remember any company meeting where there was any thoughtful discussion on, "Can people speak up on why we're not succeeding as a company?" It was almost like the engineers were pawns, the engineers were doing their thing, and higher-level people were trying to figure out <laughs> how to make this a business whereas at NeXT, which we're going to talk about soon, Steve Jobs, especially this size company-- Steve Jobs would have called everyone into the room and say, "Well, how come this isn't working, or that isn't working? Let's hash it out." That's how a functional company works. They put the cards on the table and figure out what's working and what isn't working, and that exercise, never to my knowledge-- at least, if it happened, it didn't include me.

<laughter>

Naroff: Or Brad, apparently.

<laughter>

Naroff: Again, when people come in writing checks for a small company, beware. Beware.

Hsu: Well, that's a good segue to the NeXT story. We've already talked a little bit about how Objective-C came to be used at NeXT. You mentioned William Parkhurst and Bud's original ideas about what the software should be, is there any more you want to add to that story?

Naroff: Regarding before I arrived at NeXT?

Hsu: Yeah, I guess. Or, yeah, the decision that led to [NeXT] using Objective-C.

Naroff: I think to some degree it was also born out of C being the base language, it seeming very desirable to have a dynamic message passing idiom on top of the C language, and certainly it being based on Smalltalk, which has the PARC heritage. There was also this cultural, nice match with PARC and Smalltalk and dynamism and making a small runtime that would fit on a four-megabyte 68K, that type of stuff. But they, too, were a little naïve and being sold a bill of goods that they didn't know upfront where the bodies were buried. They just didn't know. So, they decided to use it, and they then worked hard to make sure the problems were fixed. And even Steve [Jobs] was very vocal with Tom Love, so CEO to CEO about, "We took a bet on you guys, and-- " Because before I actually got hired by NeXT, I was out there doing some consulting where I was delivering a more advanced version of the translator that did better with errors, that generated better code. I was working in NeXT's business office on Deer Creek that was segregated from the engineering because it was very secretive. I wasn't allowed to see the Cube or anything. I had no idea what they were doing. They had a Sun workstation set to the side where I was working with the Sun C compiler and the Objective-C translator to demonstrate the work I had done. So, when I was out there demonstrating this work to their engineers, Steve Jobs had shown up in the entryway of the main office and had a talk with myself, Bud Tribble, Steve Stone who was the other engineer at NeXT who was my liaison, and Tom Love. And Steve [Jobs] had pretty much read Tom the riot act in terms of "We made a bet on you guys, and you're not delivering. And Steve [Naroff] is doing great stuff, but we need your whole company to be engaged and support Steve [Naroff] because we care

about the language, we care about the infrastructure, we care about the runtime, the error messages," and he went through. He sounded like an engineer. It was amazing. He was parroting what his engineers were saying and spinning it a certain way, but ultimately told Tom that, "Invest more in the language infrastructure and less in the ICPaks." And Tom listened and was respectful, and Bud was very quiet-- Bud's a very quiet person in general. He didn't say much. And I was just smiling, saying, "Here's a CEO which has a clue. My goodness. What a difference from the venture capitalist now running Stepstone. This is a guy I could see working for." So, I was just jazzed that here's someone who understands you have to make the language work before you start building on top of it, which end of the day, as Steve loved to say, isn't rocket science. It's really basic engineering. And so, that was the point in time when I think Steve [Jobs] probably told Bud and Steve Stone, "Ask Steve Naroff to come in for an interview. I don't want to deal with this company." Who knows? They probably talked about buying the company, too, but Steve is well known for targeted hiring practices. Let's just say--

<laughter>

Naroff: Call it that. It's a lot easier than buying a company, especially since he doesn't care about a lot of the other stuff, especially the ICPaks. So, yeah.

Hsu: Hmm. So then-- Talk about how the process was like joining NeXT and leaving Stepstone.

Naroff: Well, when I gave-- Well, first, I guess, it's interesting to talk about the interview. I flew out for the interview, and I actually have the interview list here. Let me see where it is. Yeah, so I had a bunch of group interviews, like I had William Parkhurst, Kevin Enderby, Joe Caporaletti who interviewed me in the morning, and then a 45-minute interview with just Steve [Jobs].

Hsu: Wow.

Naroff: And then lunch with the OS group, and then a bunch of group interviews with Avie Tevanian, Peter King, Mike DeMoney, Bud Tribble, the whole cast of characters, but all long group interviews. And when Steve came into the room, Kevin Enderby, who was a compiler-linker guy, was held over, and we were babbling about something. Kevin respected that it was Steve's time and said, "Well, Steve. I'll let you get to it." And Steve was like, "Oh, no, Kevin. You can stay." So, Kevin begrudgingly stayed for a while, while Steve proceeded to confront me.

<laughter>

Naroff: And it's funny-- Kevin probably remembers this better than I do because whenever I see Kevin he loves to mention this. Like, "Oh, man. That was tough, sitting through that." But Steve basically looked at my resume with some disdain, questioning my career, what I had done, and it was interesting. I proceeded to defend my life and talk about the work I had done and talk about the two startups I was a part of and how I wound up at Stepstone, a company he didn't respect. So, I went through that, and I saw Steve immediately relax. And I don't think I was smart enough at the time to truly know what was going on, but it didn't take long to figure it out afterwards. Steve was convinced I should be at the company

before he entered the room, yet he was very-- He made it a tough interview, and he did it because he knew I had leadership potential at the company, and he wanted to see whether I would shrivel up and die under his questioning. He wanted to see, "Can this guy defend himself? Can this guy defend his ideas? Or is he going to <laughs> shrivel up and die?" And again, given my background-- New York, played in many bands and had to deal with a wide array of people-- I just had fun telling him about all the stuff I had done. And when I got around to telling him how I thought programming was an art and how it related to music and how all that brought me to this place, he was sold <laughs>, because he really does have-- had an inner affinity for people that loved music. He loved music, and I know that he felt programming was an art. And so, I think we developed a kinship in that meeting that catapulted me to great places from that point on. I forget when Kevin-- I think Kevin stayed for 15 minutes and Steve and I held over a little bit, and then the rest of the day we had a great lunch interview with the OS team at Little Garden, which was in the day just the coolest Silicon Valley-- It was a Chinese restaurant in Palo Alto that was like a who's who, lots of great Sun engineers as Sun was coming up, and NeXT, and just listening to the conversations at all the tables was very amusing <laughs>. It was just a very incestuous place to be <laughs>. So, that was my recollection of the interview. And so, when I went back, they made me an offer, and I told Stepstone and they threatened to sue me. They did not want to bless me leaving them and going to work for Steve at NeXT. And so, I told Steve and NeXT, "Listen. This is crazy stuff. I've never-- " I was a young-- I was 25, 26, and I-- Let's see. No, 27. But I'd never been threatened with a lawsuit and I didn't have the means to defend myself, and the whole notion of them taking a lien on my modest condo that I bought with 14 percent mortgage was intimidating. So, I said, "Steve, listen. They're threatening me, and you need to solve this for me or I can't come," and he told Gary Moore, his lawyer, to deal with it. Gary dealt with it within, I don't know, a few days. And it all made sense, though, because NeXT had bought a full-source license so they had the source code. And I wasn't going to be taking any ICPak code with me. I didn't work on that. There was no intellectual property issues regarding that. So, as far as the language goes, since they bought the source code, we were really free to-- And we were going to develop our own runtime. We weren't even going to use their runtime. So, that problem got solved pretty quickly. And I think things have changed over the years. People realize companies don't own employees and that if they didn't have a source code license and I took source code out of the building, yes, that's not cool. But I'm allowed to take my knowledge of the language with me because they were working on the language. And in fact, if Stepstone was smart and Dennis Cisco, who was running the place at the time, really stepped back, he'd say, "Great. We don't want to invest in the language. Let NeXT drive the language, and we'll drive our ICPak strategy." So, it just-- People were frantic because they were having a problem running their company. There was no really grounded thinking, and if I would have told Dennis what I just told you, he would have probably looked at me like, "You're 27 and poor, and I'm old and rich. So, why are you trying to tell me what to do?" Stupid stuff like that, and that's what was also beautiful about Steve is, how much money you had or how old you were, none of that entered into his thinking. It's "What are your ideas?" And so, I think-- So, the transition was, after that scare, all went pretty well. We don't get too much into personal stuff in an interview like this, but one of the personal issues that affected my wife even more than me but was still a family thing was as we were making the plans with moving to Palo-- well, actually, Sunnyvale-- and taking the job was, my wife was pregnant. And we got a call from the doctor right before we were about to move out that he saw something on the sonogram and the pregnancy was not going in the right direction, and we lost that baby. And a traumatic event, considering you're switching jobs-- There's been so much moving, and that happens, and it really

was tough. But we got through it, and then I went into this pressure-cooker situation where I'm at NeXT, I start learning about the hardware and the strategy and the fact that-- I think I started in August and the--

Hsu: Of what year?

Naroff: Eighty-eight.

Hsu: Eighty-eight?

Naroff: Yeah. The interview was in May. Maybe I started-- I don't know how many months, but some number of months past. Close enough. I think it was August that I started there, because cross-country moves and the pregnancy issue and stuff. But we had the introduction of the Cube in October, October 12th, I think, of '88. So, we were in the bomb run. That was a big NeXT term. I think Bud coined it, but who knows. Bud's always like, "You're in the bomb run for the release." So, here I'm new, don't have a product yet, so many pieces on the floor. It was a tumultuous time, but a really exciting time because we were-- NeXT was the talk of the town, then. It's actually interesting that this is October 8th. So, Steve passed last week in 2011, and the intro was the end of this week. So, it's interesting that this is happening between those two pretty big events, in my life, at least. But let's see. Well--

Hsu: Yeah. So, what was your first position, and what did you work on when you first got there?

Naroff: So, I was an engineer who worked for a fellow named Mike DeMoney who was in the operating system group. Avie Tevanian had hired on a couple months before me. We're very close. He was on the interview schedule, but he was pretty new. He was not running anything. He was Mr. Mach and was doing OS work. So, I was reporting to Mike DeMoney, who was the manager for the OS group, and it was a big, open room with-- I was thrown in with the OS group. So, there was no tools group. Steve Stone, who was basically hacking a compiler prior to hiring me, was trained as an OS guy but was just a jack of all trades, and he loved it and was driving this hire because he was just dying to get out of the compiler business.

<laughter>

Naroff: And his clients throughout NeXT Software were also dying for him to get out of the compiler business because they had lots of bugs that were on their list, and Steve [Stone] was an OS guy and he had lots of stuff going on helping out with the OS and stuff. Doing compiler work in your spare time when you aren't a compiler guy and trying to help people, it's tough. So, I was-- The very first thing was just fixing bugs in the Stepstone translator that I had done as a Stepstone employee and making sure the integration with, at the time, the GNU compiler-- See, that was totally new to me. I had no idea that they were using Richard Stallman's Free Software Foundation product, which was also very new. The 1.0 of GCC happened in 1987, and this was '88. So, it was a year out of the chute, out of the 1.0 release, and there were bugs in it, too. So, bugs in the Stepstone translator, bugs in the early versions of GCC, and that was, I'd say, at least two to four weeks of my first month there. And it's interesting, after two to three to four weeks of bug fixing and concentrated helping everyone, I know the person who was working on

the Workspace, the “Finder” at the time, Chris Franklin, I remember sending me a fairly public email that was copied to most of Software saying how much he thanked me for helping fix these bugs and how it's really helping their productivity. It was such an open environment that simple things like making sure the compiler worked was like I was a rock star just because they were getting stuff done that they couldn't get done before, and that was great. It made me feel-- It motivated me. I felt like I was needed. I went from an environment where I was begrudgingly let loose to do anything interesting to an environment where just fixing bugs made people giddy.

<laughter>

Naroff: It was-- So, honestly, really soon after that they raised my salary and treated me really well because they just were happy. And I think we've talked offline-- NeXT had an open salary and stock structure where people, when they were happy with other people or not happy, some people-- It really wasn't referenced often, I don't think. But in this case, I think Chris or others went and saw what I was being paid and my stock position and very soon after, because it was an open salary structure, my clients lobbied for me to get a bump. And that was also unheard of. I had never experienced anything like that. I was-- So, it was honestly the best environment and culture I had ever experienced. So, this just led to me being even that more devoted to doing crazy great things. I'm like, "Wow, it's being recognized." And so, I credit a lot of the cool stuff I did over the years with providing an environment that was nurturing and appreciative of things, which is surprisingly hard to find--

<laughter>

Naroff: -- I think in any field. I was really lucky, really fortunate. So, other than that, then we had to draw plans for, what is next? When are we going to not be dependent on their runtime? So, I started working on a replacement runtime that was optimized for Mach and optimized for their-- And I had-- The only two people working on compilers then was me and Kevin Enderby. He worked on assemblers, linkers, loaders, and I worked on the frontend and the middle and the integration with GCC. And so, I had had ideas for what the linker could do for me to make Objective-C better, and Mach-O [file format] had very interesting features for organizing data, organizing the metadata, and we developed really interesting system-specific, Mach-specific ways to optimize the metadata, so the paging was minimized and that all the stuff was organized and accessible really easily. So, that was fun, and Kevin did a lot of that work. And just more and more, owning a linker for a language like Objective-C is really important because even though it's a dynamic language and you could just defer to runtime for so much, because we were running on modest machines with modest memory, devising creative ways to distribute the workload, get the compiler and linker to do as much as possible and deferring as little as possible to runtime was really important, and it took a fair amount of work to do that well. So, we weren't doing that much with evolving the language. We were evolving the implementation. Now, one notable exception which happened fairly early on was Categories. Jean-Marie Hullot who was working on Interface Builder-- Again, not in any formal team, but just another part of the software group. He had come to me and talked to me about what magic he was doing to make Interface Builder work. And he was using something called “Pose As,” which was a runtime hack to basically take a class and interject another class ahead of it in the metadata so that you could basically spoof the runtime into executing code. And while that was cool, I decided that it

needed linguistic support to support Interface Builder because Interface Builder was really the crown jewel, and making sure it didn't depend on hacks but depended on language features explicitly is sort of important. So, we developed Categories for that.

The interview was resumed after a break for lunch

Hsu: Yeah, we were talking about Categories and Interface Builder.

Naroff: Yeah, so Categories are a way to add a set of methods to an existing class. There's a simple syntax to do it, and at one time, the class's methods and the category methods are unified, so that the message dispatcher can find them. And it worked out great. People loved them. It's an interesting example, going back to-- I mentioned briefly, before, the "Object-oriented Programming is Bad" video that's popular out there, and how classes are almost too structured. Categories are an example that are loosening that structure, and I think it's an example of adapting classes in a more useful way without being dogmatic about it. There are-- it's not a perfect world, in that if you have two categories with two methods that are claiming the same implementation, there are ambiguities at runtime that can be difficult to resolve. So it's-- you have similar problems to multiple inheritance, or the confusions that some people have with multiple inheritance, precedence of multiple ancestors, but it-- in practice, it worked, and people liked it, and it made things much easier than using the "Pose As" hack. So, again, a pragmatic-- I think one of the things that drove me was doing something that made things better and useful, and not evaluating it as linguistically perfect, but moving the ball forward. Because, some of these problems don't have perfect solutions. And if you can solve the 98% case, do it, rather than depend on a hack. Other people might make other decisions, I mean, I'm not saying it's right. But I know the reaction to Categories was very positive, and I think you've used the feature, right?

Hsu: Yeah. And they're-- today they're called-

Naroff: Extensions?

Hsu: Extensions, I think, yeah. Yeah, I don't think they use the name Categories anymore, but it's still there, yeah.

Naroff: Yeah, and one of the inspirations was Common Lisp. I mean, Common Lisp was evolved from Lisp to be an object-oriented system, and Common Lisp classes are totally open. Anyone can add a method to a class. And that was sort of proof that, you know, it works, even if it's not perfect.

Hsu: Right. And was Jean-Marie-- he had-- I heard that Interface Builder had originally been developed in Lisp. Is that correct?

Naroff: Yeah, I think he did the first Interface Builder, for him, in Lisp. And I forget where he did it. It wasn't at NeXT, but it's the implementation that I think he showed Steve, where Steve basically got interested in Jean-Marie, and said, "Come on board and do it in the infrastructure we're developing." And so he did. And Jean-Marie was just a real pro, you know? You can imagine having some people from that

background that would lobby, “Oh, we have to move to Lisp, because my tool is in Lisp.” There are people out there like that. But he was so pragmatic, and so easy to work with. Just a wonderful guy.

Hsu: I mean, that’s a really good example of how the language was modified to support the application, or the tool.

Naroff: Totally.

Hsu: Rather than the other way around.

Naroff: Totally. And one of the reasons-- like people sometimes wonder, why didn't NeXT choose C++? Because as-- well, we’re going to talk about soon, we had to support that language. And supporting it, for other people who are using it, is different than basing your crown jewels on it. The reason we didn't use C++ is, simply put, it didn't have dynamic features that we needed. And we certainly weren’t going to wait around for AT&T and Bjarne Stroustrup to service our particular needs. And we knew we weren’t big enough to influence them. So one of the things that Steve [Jobs] and the rest of the engineers who chose Objective-C were really smart about, whether this was absolutely intentional or not I can’t say-- I think it-- having talked to Steve about it, I think from Steve’s perspective it was intentional, is that he knew they had more control over Objective-C than they would have C++. So, you know, control and evolution was really important, and it just started out where they wanted to be. I mean, C++ being a better C, with all of its static sort of view on life, just wasn’t that interesting to the folks at NeXT at the time. Yeah.

Hsu: Yeah, I want to-- I mean, you mentioned dynamic. Why was having a dynamic language and runtime so important to NeXT?

Naroff: Well, there’s no doubt that it starts at Interface Builder, in that, when you have a palette of objects, part of that vision was having a custom palette of objects. And as soon as you talk about custom palettes of ObjectWare, or ICPaks, you're talking about dynamic, right? Because the tool is not going to be statically aware of everything that can possibly be a part of it. I mean, you have the same issue-- let’s face it, like a PDF file. You have many different applications that can operate on the PDF file. Well, when you have a menu pic that populates with all the tools that know how to manipulate the PDF, that’s dynamic, right? And so the whole notion of extending the data types that either Interface Builder or Finder or Workspace, I guess we called it [back then]-- you know, we knew that the system, for the next generation, was going to be dynamic. Dynamic is also-- or, I think, from our perspective, rather than-- the rest of the world viewed dynamic as inefficient. And I think the more we learned, the more we realized, no, dynamic is actually more efficient from many perspectives. Maybe not all, but many. An example is, imagine you have all these kits, with many classes. Well, one model is, every kit you link against, that there is some overhead on a per-kit basis. So you have three kits, each having 100 or 1000 objects, and that, independent of use, you're going to do some sort of initialization in anticipation of use. Well, that’s how a static system would work. It’s like, oh, you're linking against this library. It has to be prepared for its use. Well, one of the things I did in this early reimplementation of the runtime was, all class initialization was lazy. So if you linked against a kit with 1000 objects, and you used three, the only three that would get initialized and prepared for use would be the ones you used. So that’s dynamic. You know, the only

way to scale is dynamic. It's really the only way to scale anything. And the way we grew up, with slow processors and slow compilers, there really was this tendency that, the machine is so feeble, that let's do everything we can, or learn everything we can about this piece of software, when we have the opportunity. You don't want to defer anything. But we learned, over time, that that's not practical. As the system grows, that's not practical. Same thing with MAINSAIL. They did everything lazily. So if you linked against 100 modules, the only modules that got brought into memory and used were the ones you used. So there's just so many examples where the other-- actually, another point is, most of the early-- well, all of the early Unix systems didn't support the notion of a shared library, a dynamic shared library. They were statically linked. And Domain, the AEGIS operating system for Apollo, had a really great dynamic shared library scheme. Amazing. And they influenced the industry, because the Unix guys said, "Well, clearly, that's better." And why did Apollo have that? Because these CAE/CAD (Computer-Aided Engineering, Computer-Aided Design) applications were being ported to it, and they were driven to do, I believe, architectures that scale better. So it's interesting. When I got to NeXT, they had a very feeble implementation of shared libraries that was based on UNIX System V. It was-- it was position-dependent shared libraries. What that means is, developers actually had a list of symbols with the exact address where that symbol would be in memory. And the linker helped manage this. I don't-- I forget all the details, but it was not position-independent code. It was not the compilers generating code that it doesn't depend on where this resides. So that was another thing I lobbied for, which was to move to position-independent shared libraries, more like what I had seen on Apollo, or other modern systems of the day. And we, you know, I made that a priority, and we did that before the 1.0 release we had-- I think it was done before the 1.0 release. But it was definitely something that I lobbied for.

Hsu: So you were the primary-- I guess the Directly Responsible Individual, as they would say-- I don't know if they used that term then, but you were the primary person responsible for the Objective-C runtime at NeXT?

Naroff: Yeah, oh yeah. Oh yeah. No one else was working on it. When we hired Bertrand [Serlet], I'd say within four to six months of me hiring on, Bertrand had great interest in the runtime, and, you know, worked with me and voiced opinions. And, you know, he had great experience and great insight, and he-- you know, I'm sure he did influence stuff I was doing, but it wasn't his primary job, and he wasn't in the compiler group. But he was interested, and certainly, later on, other people got interested and offered insight, which, again, was really cool. I love people critiquing what it is I was working on. I'm not shy about, yeah, that's-- I never thought about that type of thing. Again, I think my music background lends itself to that, because, you know, music is one of those domains where-- musicians are always paying tribute to other musicians by using their riffs and using their ideas. And it's considered goodness. In computer science and engineering, it's not always considered goodness, depending on your background. I've worked with people who come from a very academic background, who sometimes only want to use the ideas they conceive of. And if other people are conceiving of them, even though it might be a better idea, they're not as prone to take it. And that, in my experience, is not the best path. So, Bertrand, as I said, he and I talked about the best hash schemes, about reentrancy, there were things he noticed that I'm sure we improved, based on his feedback. And the runtime was small. That was one of my-- one of my big things is, if I would have let everyone at NeXT have their pet feature, or have everyone hack away at it, it could have ended up differently. But I was pretty strict about only letting really important stuff in.

And it was easy to justify that at the time, since we were really behind the 8-ball in so many things. Because we still didn't have a GNU compiler that knew-- that understood it natively. I was-- that was like the next thing. First it was getting the translator and a core runtime, and then hacking the compiler directly, so we would have no preprocessor.

Hsu: Let's actually talk about that, next. But before we get into that, why do you think that GCC was selected in the first place, to be the compiler? And was there any culture clash there? I mean, NeXT was doing a sort of proprietary system. Steve Jobs was known for proprietary systems. Was there a cultural mishmash, using Stallman's stuff?

Naroff: I didn't feel it, if there was. As for why they chose GCC, I really don't know, except Steve Stone, who was in the trenches, I simply think thought it was some good work, and that they were kicking the tires. And they were kicking the tires, let's face it, with Objective-C too. And so the issue was, will they converge? Will this language converge enough for us to ship a product that isn't shitty, and will the compiler converge? The compiler was 1.0, and it was pretty respectable, but, you know, it still needed work. And, I think, on both fronts, that was my job, to make those two things shine for them. And it all worked out. I mean, Richard Stallman was an amazing person to work with. I can't say I subscribe to all his politics, regarding free software and patents. You know, I have patents, so Richard probably doesn't like me, because if you-- if you're someone who's name is on a patent, or someone who doesn't universally give away anything you've ever done, you're bad. I mean, he's hardcore. So-- but the fact that he's hardcore, and politically I didn't necessarily buy into everything, didn't affect me, because he was a brilliant programmer. He was a brilliant leader of GCC. And though GCC might not have been the most beautiful compiler ever written, from my perspective, it was really, really good for the time. It was very good for the time.

Hsu: And, I mean, it seems like the fact that you could get the source and modify it was a benefit to NeXT, because then you could add Objective-C.

Naroff: Oh my god, I could add Objective-C. Yeah. And that's-- I did that, and it, you know, was not that much work. I mean, basically, you modify the-- he had a tool called Bison, which was his take on Yacc, which was a compiler compiler. So I go into the rules, and I add the Objective-C grammar, and add the actions. The trickiest part about doing that project is not the syntax, you know? Recognizing syntax was fairly straightforward. It's-- that compiler was generating parse trees, generating data structures, complex, composite data structures that needed to be extended to do Objective-C. And you needed to, in many cases-- because, under the covers, I was still generating C code, but rather than generating text, which was C, I was generating parse trees. So I had to synthesize Stallman tree nodes to make his compiler happy. And that was, you know, the toughest part of the job. But once you did that, it just all worked. So his parse trees weren't all that necessarily clean or beautiful, but they were very highly functional. They made sense, once you had, you know, once you were past the learning curve, because it's a complex, fairly big compiler. And, yeah. That part of the job, making GCC do Objective-C, wasn't that big a deal. And Richard was very helpful, and fixed bugs when I found them. Because, clearly, when you're-- just like the-- I guess, schtick I had before, about writing apps to prove a library, when you add a front end, you're proving the parse trees. And you're saying, how come this parse tree doesn't fit for what I'm doing? Let's

modify-- is it the right thing to modify the parse tree? Or to have a separate data structure, to do this, whatever that is? And so getting his architectural buy-in was very important, because I wasn't, you know-- but getting back to part of the question you asked, which, I think, now is coming to mind, is, the one mismatch of culture, so to speak, was, Richard wanted everyone to be doing this work in the open, right? He wanted all the FSF, GCC development to, every time I was doing anything to Objective-C, to be done, integrated into the mainline, and done in clear view. But, because NeXT and most companies operate on product timelines, it made much more sense for us to say, "We want to work on Objective-C atop GCC until we ship our, let's say, 1.0 version. And when we ship our 1.0 version, we will then take all of our changes and work with you on integrating them." Right? Now, Richard wasn't in love with that. Like, for a while, people-- oh, NeXT isn't being good citizens. They're not contributing back to-- it was just a timing issue, a pragmatic issue. It wasn't any religious issue. We always wanted to give all of our work back to FSF. A more, I'd say-- another obvious example of the discord between what NeXT was doing and what, let's say, Free Software [Foundation] does, was, if NeXT was going to introduce a new hardware platform, with a new processor, well, you know, NeXT was fairly secretive. If we give back, let's say, PowerPC changes, or whatever the processor is, that's going to be a big red flag that, oh, NeXT is working on PowerPC, right? So-- but see, those, to me, those problems, if they're really problems, is more a function of a product company versus an open-ended compiler development branch. So I don't know-- Richard and I never really had a lot of personal problems. I know he's someone that still likes to badmouth NeXT and Apple. I think-- I don't know if he has any fond memories, like I do, of working with him. I doubt he does. He's not a warm and fuzzy type of guy. But I have nothing but good things to say about his contribution, not only to the compiler world, but to open software. Well-- oh, I used the term "open." He hates that. Free software, sorry, Richard. And so many other things. So-- in fact, one time, he visited, and he-- we were going to integrate and show him some of the work, to get feedback, because, again, we weren't doing it in the open. So he came to NeXT and worked with me. And, you know, he-- I don't think he and Steve ever really hit it off. The one time they met, I don't even think I was in the room. You know, they duked it out by themselves. So I never-- I don't have any Richard/Steve stories. But Richard, he was a quirky dude. Like when he was coming to see us, he was adamant about staying at my place. And my wife and I had a really small apartment in Sunnyvale, and I knew she wouldn't get Richard, so I said, "Richard, we'll put you up in a hotel, you know?" He's such a funny guy. And then, later on, in the early '90s, Mark Wagner, who's a guy I worked with, we went to MIT to talk to him about something. I forget the details, even. But he used to sleep in his office, you know? On a little foam mattress. And then he'd go into the men's room and do whatever he was doing, brushing his teeth. He lived at MIT. Such a character. Such a character. And I think, now, he mainly is involved in politics. The politics of free software. And I don't really see that things have changed much since the NeXT days. Do you notice any change in the whole free software movement?

Hsu: I don't think I would really be able to answer that question.

Naroff: Oh, I mean, I respect that he's still fighting the fight. But I don't know if you plan on asking about this, but another misconception is-- well, I'll talk about that later.

Hsu: Oh, okay. Okay. Sure. Remind me, if I don't end up asking, what you think-

Naroff: I'll remember. Yeah.

Hsu: Okay. Was there any conflict over the fact that the Objective-C runtime was not open source, or free software, sorry-- the runtime was always proprietary.

Naroff: It was. It was. We documented the interface that the compiler depended on. And we felt that documenting that interface was our responsibility, so that someone else in the free software world could write their own runtime, and other people did. Someone I hired as an intern, I forget the exact year, but Kresten Thorup was a Danish student who was just very talented. We hired him, and I believe he eventually contributed an Objective-C runtime. And, again, Richard, I'm sure, wanted-- but the runtime was a low-level piece of the operating system. And there were issues of the whole copyleft, that if you link in this type of software, into applications, which we were, or it was in our system libraries, would it infect the whole system? And we-- our lawyers just didn't want to deal with that. So, I think, Steve and NeXT really did do more than any other company at its time, certainly more than Microsoft was doing at the time, with embracing, in limited ways, open source ware-- free software, and the open source development model, where you're collaborating with other people. And, as you know, we [Apple] followed through on that with LLVM and Clang, which we'll talk about later. And we did it with Safari, with WebKit, and there are lots of examples. I think Mach 2 was obviously taken from CMU, and I don't think that open source really flourished. I don't know-- I think, basically, NeXT did have a lot of proprietary mods that never made it back to CMU. I'm not-- it's not my domain. But I do know that we dipped our toe pretty heavily, and, I think, deserve some credit for doing that, because Apple was-- well, not Apple, NeXT. I think Steve thought that, for the system plumbing and operating system and compilers, this was a very valid reason to leverage open source, because at the end of the day, they got an Objective-C compiler. I mean, and later on, they got an Objective-C++ compiler, which I did.

Hsu: Yeah. Well, actually, that's a good segue to talk about that. So why was it important to add C++ support to Objective-C?

Naroff: So we had three big third-party software developers that were heavily into C++. We had Lotus, who were developing Improv. They were probably the most important. We had the Adobe team, with Photoshop. They were big clients. And lastly, there were some-- even some Pixar stuff on the radar screen, that was being done in C++. Three pretty important companies.

Hsu: One of which was owned by Steve.

Naroff: Yeah. And Lotus, in particular, was really unhappy with what they had to do, which the workaround, without the work I did, was to create a C-function that they could call out, in an Objective-C file, obviously, that would then call in to the Objective-C kit, to get their job done. Creating stubs for every interaction with the kit is very time-consuming, right? And a maintenance burden. So it was immediately obvious to me that this needed to be solved or else their lives would be miserable, and as a side effect, my life would be miserable. Because I was, you know, becoming the tools guy. And so Michael Tiemann had just, in, god, I think it was '89, early '89, released the first GCC C++ compiler. Michael was a Stanford grad, and Michael Tiemann, John Gilmore, and David Henkel-Wallace were starting Cygnus Software, on

University Ave. And I briefly touched base with Michael, to get the lowdown on the state of the compiler, and he was pretty optimistic that, though it wasn't finished, it was compiling quite a bit of stuff from the C++ perspective. So, because he gave me enough positive feedback about his work, and, again, I didn't really know Michael, but he seemed really smart, and I trusted him, the little bit I knew him, I decided to try, on a weekend hack, to take his work and add my Objective-C work to it. And-- because, same problem as I was discussing before. Where Stallman's C-based nodes weren't cutting it for C++, so Michael-- you know, it wasn't developed as an extensible toolkit. It was developed as a compiler. So then, you know, Michael's parse tree mods and my parse tree mods would have to be reconciled. So that was part of the problem. So I got Michael's work in a weekend hack, got it pretty far, and was really pretty blown away that-- I mean, the grammar was complaining about this, that or the other thing, because compiler compilers, when they're generating their tables and nodes, can detect ambiguities. And those ambiguities are sometimes pretty hard to discern. But I basically didn't let it bother me, and I just went heads down and continued. Long story short, after a couple weekend hack fests, because I didn't have time during my normal hours at NeXT to work on C++, got it far enough that I shipped Adobe-- not Adobe, I shipped Lotus the compiler, and they were like giddy that, "oh my god, this is like doing a lot of good for us. And please finish this." And that work ended up being used by them, and then the Photoshop team, and Pixar, and it became really popular, because people could-- just like they bring over their C code and start using Objective-C immediately, if you've used C++ as the next coming of C, then it makes total sense. Because what I wasn't doing, which some people initially thought could be the design point, is to merge the object models, right? Like allow a C++ class to coexist with an Objective-C class. For example, you know, that would allow potentially sub-classing in Objective-C, a C++ class. I thought that was just idiotic, and even if someone was lobbying hard for it, I would just say, "Listen. That's not the design point. It just doesn't make sense." Now, later on, it's interesting. People tried to make, with SOM, there are other, COM and SOM, there were other object models that tried to basically make C++ a little more dynamic. But I just never thought it made sense, unless C++ authors thought it made sense. So, does that make sense, how that evolved?

Hsu: Yeah, because-- like the reason you didn't think it made sense is because C++ is fundamentally a static-

Naroff: Fundamentally C. It's just C. I mean, they-- yes. It's basically an improved structure, right, with the ability to add methods. But it is very static, and not doing anything dynamically. And it would just-- that would be a research project. That would be a research project. So-- but one of the things, let's see, I wanted to-- god, do I have it here? I'll find it.

Hsu: But you just wanted to do the pragmatic thing, to make it work.

Naroff: I wanted to solve their problem.

Hsu: Right.

Naroff: Because really, solving the other problem was a tremendous effort. So-- and, again, it really made no sense. It really didn't make sense. So, wow, I could solve their problem in two weekends, get the

compiler 80%, 85%, then finish it off. And, eventually, Michael finished his work. We completed it. And it ended up being pretty robust. But even in the day-- like what I'm trying to find here is-- oh yeah. So here's like-- Lotus ships me a letter. They said, "As a token of our appreciation for your efforts developing Objective-C++, been authorized to present you on behalf of the Lotus Back Bay team, the coveted code talks, bullshit walks t-shirt." Okay? "The Objective-C++ compiler correctly compiles all our code, allows one to use all the features of both Objective-C and C++ in the same file, and was finished and delivered before we expected it." And they go on to say, "The code talks shirt is only given out when a team member has finished something before people thought anyone could have finished it, or has done something no one thought could be done. Or, in this case, both." So, you know, it was just a really big deal, to enable them. And Michael Tiemann's work, open source, wouldn't have been done without it. Imagine we were dealing with a proprietary C [compiler]-- imagine we were dealing with AT&T. Would have never happened. So, you know, I wish I could say, in retrospect, we were all so brilliant that, you know, all this great stuff worked, because we're such great planners. There was so much serendipity here. We didn't control Michael Tiemann. He came out of the woodwork. And doing what he did is serious work. C++ is a serious language, and it takes a very special person to do what he did. And we leveraged it.

Hsu: I think I wrote here somewhere that there were plans, at one point, to support Lisp in the [Objective-C] runtime? Is that correct?

Naroff: No.

Hsu: No. Okay. So, okay, never-- I thought I saw something about Franz Lisp in one of the documents you sent.

Naroff: Oh, wait. There was a time when Franz Lisp, they brought over their system, and they wanted to interoperate with the runtime. That did happen. But that wasn't strategic. In other words, the Objective-C runtime is a library like any library, and if someone said, "We want to be able to--" I think we even did Objective Fortran, I think-- but we didn't do it. We just said, "Oh, if you're a Fortran compiler writer and you want to send messages into the runtime, here's how you do it." Again, another beauty of dynamic, right? If it wasn't dynamic and only the compiler was able to do the magic, then you couldn't easily do it, as easily do it. But most of these vendors have figured out how to call into C. It was a standard, so...

Hsu: So earlier you mentioned Bertrand coming in. So you had helped hire Bertrand into NeXT, or were you involved?

Naroff: I was part of the interview team and was really impressed, and I wasn't inter-- I think I was managing a small team. It was me, Kevin Enderby, Matt Self on compiler, and Dave Moore, who's now a Google employee at-- on the debugger. So we had a four-person team, I think at the time I had started managing.

Hsu: Oh. So you had officially become the Dev Tools team?

Naroff: Yes.

Hsu: The Development Tools team at that point?

Naroff: Yes. I'd say by, yeah, by late '89, early '90, I was-- I did that. I had taken over, and yeah. It's interesting. There were some discussions about, in NeXT, where to go with the tools, and Bertrand had written an e-mail about really feeling strongly that the performance analysis tools were insufficient.

Hsu: This is after he joined?

Naroff: Yes. Right after he joined. He wrote an e-mail in February of '89, which was sort of after the first, my first six months of being in the bomb run, when I was coming up for air, and he had been hired, and he had voiced some pretty strong opinions about his need, personal need, for doing performance analysis, and he had done it in an e-mail to, you know, the Software group, and it's interesting, you know, I had replied to him and copied the same big cast of characters. Still have this e-mail, and I told him, "I agree, [Unix] gprof is totally inadequate," and I told him what I don't like about it, which confirmed what he didn't like about it, and-- but I went on to say, "We're in the bomb run for release. We can't-- we need to help you guys help yourselves, even if it's internal tools we develop." Having a tools team, given all the other stuff that was happening at the time, to sort of drop what we were doing and major in performance analysis tools, was not what we needed to do, and I list four things that-- let's see. Was it four? Five things. That we needed a reliable debugger with a good interface. GDB wasn't cutting it. Debugging was more critical to people than performance analysis at that time. Plus, the operating system has more, a very unique set of performance problems, right, and so I went on to say, "Dynamic loading is important, we need that. We need to upgrade to position independent shared libraries. We need better browsers for development," you know, you can't reuse something if you can't find it, "and replace [Unix] Make," that it's just, you know, not appropriate, and Steve Jobs read it and, you know, sent out immediately, "I support your plan," and I think that's the time, I mean, I might've already been a manager. I forget. But it strengthened his belief that this is the right guy managing the team, because he's all about focus. So he was seeing in me, you know, rather than get distracted, as long as you help the OS guys figure out their performance... So it wasn't like I was telling Bertrand, "Go away, we don't care about your performance problems." It's just that as the tools group, we have to distinguish between what we're doing for third parties and what we're doing for internal people, and so that-- that's probably more than you asked for, but... <laughs>

Hsu: Oh, no, no, that's good. Because I was going to get to talking about more… See. Yeah, and I asked, I was going to ask you about going, becoming a manager, which you touched on. Let's talk about maybe the development of the developer tools over time. You worked on things like, what, the ObjectBrowser, AppInspector, ProcessMonitor, MallocDebug.

Naroff: The team did.

Hsu: The team. Or the team did.

Naroff: Yes.

Hsu: Right.

Naroff: Yeah. Dave Moore was working on the debugger, and one of the things he did was integrate with Edit so that you can have the ability to browse the instances in line in the editor, some of that integration.

Hsu: So Edit was the text editor?

Naroff: Yeah.

Hsu: Right. So all the coding was done in Edit?

Naroff: Yes. So you code in Edit, and it was trying to bring some of the debugger niceties away from, like, having to go to a command line debugger and have a UI that is in the scope of the editor. That work was, when compared with IDEs that followed, was modest, but at the time it was really useful. I mean, when you have a four-person team and you're in the bomb run for so many different things, we had to be modest. So I think a lot of those things were interesting for the time, but not maybe by today's standards, some of that stuff might look pretty basic.

Hsu: Well, I think it might be useful to sort of describe what it was like developing an app on NeXTSTEP at that point in time, when there was no IDE yet. What were the set of tools that one was working with?

Naroff: You know--

Hsu: Was it part, like, mostly command line? Were some--

Naroff: There-- some command-- see, a lot of the internal developers, since they're OS people were, like, for instance, lot of them liked Emacs. So the benefit, you know, if we were to wave a wand and have an integrated Xcode-like development environment in that point and time, interestingly, that wouldn't have been embraced by a lot of the OS guys, because they liked Emacs. And, you know, even though they acknowledge why an Integrated Development Environment that doesn't depend on a command line is important from a higher-level standpoint, that's not what they wanted. So there was always this tension between steps we can take to make internal people happy versus people who aren't internal who don't have a deep UNIX background, right, and that-- that was true for so many years. It was a really tricky balancing act. Make too. Make's an example where it's a very file-based way of maintaining programs but having a language-based way where if you have a function with a hundred methods and only one changed, why should you have to compile all of them? Now, later on we did lots of interesting work, some of which is patented with doing more incremental forms of compilation, but back in the NeXT days, we didn't have the wherewithal because we didn't have the resources and we were in the bomb run always. We were always-- that's the one thing that was-- it was good and bad. I mean, it was good in that our priorities were always not that difficult <laughs> because you really didn't have that much freedom. So that's why I always, like, would try and pick, like, let's take the last discussion about

Objective-C++. That helped no one internally, right? So from the OS guy's perspective or other people's perspective, they're like, "Why are you guys working on that? Who cares about Adobe? Who cares about--" right? So I was always the manager who had to, like, have the thick skin to just lobby, "Well, here's why we're doing it," and we have four people, <laughs> so-- and lots and lots of code to evolve and test and manage. So what were the other tools you were talking about?

Hsu: ProcessMonitor, MallocDebug, AppInspector. I guess that's-- actually, I wanted to move to-- you mentioned Make. You developed something called DevKit?

Naroff: Right.

Hsu: Could you talk about that and explain what that is?

Naroff: Yeah. So in '91-ish, '91 to '92, we had more breathing room, because we had shipped several-- we had shipped lots of stuff, and we really felt like we could start innovating a little more. So one of the things we worked on was precompiled headers, because we had these kits with thousands of header files, and it made the compiler slow and it made it impossible to find stuff. Okay. So what other IDEs had done in the time, like, LightSpeed C and CodeWarrior, other compilers out there, is they had precompiled headers in a very limited form to make the compiler faster, and to be a little bit more specific, the C preprocessor is compiler writers' worst nightmare. The C preprocessor does arbitrary textual substitution. It gives you a level of flexibility to manipulate files that most people don't need. Right. But because it's so powerful, it makes developing a feature like precompiled headers really difficult. Because some people can use this crazy stuff like, "Oh, if I'm in this context, define this structure," or, "If I'm in this context, else, define this structure," with different structure bodies or fields, okay. Gives you some crazy capabilities, and... But we were the OS vendor. We owned our header files, for the most part. We could place constraints on what those header files could contain, and we wanted to go further than other systems had gone and allow you to precompile your own headers, system headers, arbitrary, and we wanted our browsers to drive off the same database. So if you had a precompiled header for thousands of object definitions and method definitions, the human should be able to find that definition as fast as the compiler could find it. So I just love that idea and I hired someone named Andy Lippman, who worked on it with me. I did the DevKit work, which was a reusable parser that was independent of GCC. It was a full-fledged, yet another Objective-C parser.

Hsu: <laughs>

Naroff: Okay? But it had the property, when I call it reusable, it did not by default do anything but parse the program. Then you could add your semantic actions and, like, if the semantic action was make the compiler faster, it would do one thing. If the semantic action was, "Go find this in a header file because I'm a browser, a graphical browser," do another thing, and for its time it was really an amazing idea, okay, and it was one of those things that I had written three or four compilers before that, so it was an evolution of, "Oh, my God. I've written all this stuff. Let's factor out what's important." Yet another example of would I have ever-- people don't arrive at that because most people never get the opportunity to write one compiler, let alone four, right, so-- and in fact, that design, I ended up implementing in Clang and LLVM

later on, almost verbatim, because it was just right. You get to a point sometimes, and that's where the-- getting back to ICPaks, when you get to that point where something you've done, you've done it four times and you know you're at a place where there's very little to ring out of it, improvement, that's when it's time to become a software IC. Right. So to speak. So anyway, now, back to-- so we did that, and HeaderViewer, which was an app I wrote, that was on top of DevKit, was loved by people, and the precompiled header stuff worked great, and just to-- there's a paper out there that I have that you could read that Andy wrote. He got it accepted by the ACM and it was published, and it describes this, but I could describe it very quickly. It was a really cool idea where, let's say your-- you got HelloWorld, calls printf, but you have an import or an include that includes thousands, tens of thousands of declarations. What this implementation would do is do a dependency graph of all the declarations you need to correctly compile that module, which in this case is, let's say, main printf. So it would just pluck out printf, put it-- rather than put all the other system functions that surround printf, it would compute the fine-grained dependency graph, like, it would go to the function prototype, look at its arguments, unwind recursively and figure out the complete, fine-grained dependency graph and do it really fast, and it was really a really cool project. So DevKit enabled that. What was the other part of that?

Hsu: No, I think that's fine.

Naroff: That was it? Okay.

Hsu: Yeah, I think that's it. We're sort of working up to it, but I think the next thing I want to talk about is the creation of the IDE itself, Project Builder. How did that come about?

Naroff: Dave Moore drove that. Dave was a really talented engineer who started working on GDB, and when he was able to dig himself out of fixing GDB bugs and getting GDB to work, he conceived of Project Builder along with, obviously, input from others, but he drove it and created it, and, you know, was really good at carving out enough that he can get it done in the release schedule we had, and like so many things we did, it was beautiful along some spectrum but not complete along other spectrums. But I found when we went to, like, NeXT developer conferences and other forums, I always felt like the developers were on our side and the developers were really happy with the choices we had made. You're always going to have some radicals who don't understand it's a four-person, five-person [team]. I forget whether it was four or five people at the time, but in general I think most people sought evolutionary progress from release to release and were really happy with what they were getting. Like, we didn't have, for instance, at that time, like, even though we had DevKit, we didn't have the support of doing code sense or like when you're in the editor, to complete on methods. You know, we could've had it if we had more resources. But doing that right is actually not trivial. It might-- it's not that hard from the language perspective, but making sure the UI is zippy and fresh. So we weren't getting a tremendous upheaval about code sense. Once, later on, when it became a standard feature of Visual Studio, as time evolved and Microsoft was investing so much money in tools, then people started to compare with other development environments. But I think people said, "Oh, Interface Builder rocks. The language has been getting better every release. This HeaderViewer thing's really cool and integrated with RTF files," so the documentation was in Rich Text Form, that too was pretty rare for the time period. So I think people felt like they were productive overall and weren't just measuring us by some industry standard yardstick. So I

have nothing but-- when I look back and, let's say, read appraisals of what I had done and some of my managers' feedback, it was all, it seemed to all fit pretty well, and I really don't feel like through the NeXT years there was any big discord about what was going on. The only problem I had was just we always seemed starved for resources, because we were changing products and adding processors and, you know, the other aspect of this is making sure the compiler generates great code. You have companies like IBM that have like 200 people working on back ends and, you know, we had one person. Really half a person to focus on the back end. So we were somewhat dependent on the Free Software Foundation to make sure the back end was pretty good, and it was. I mean, when you have a portable compiler or a compiler that's being designed for many languages and many processors, there's always going to be some concession. But I always felt it was pretty, pretty good, and people would submit bugs and sure, sometimes it would totally miss an optimization and we would work with Richard and others and make sure it happened. I don't think code optimization was ever a roadblock. It was more the innovation we needed to do with Objective-C. That was our baby, so we had to make it work. They weren't going to help us on that.

Hsu: And Project Builder came out what year?

Naroff: Oh, wow. I think it was '91. But I have documentation that will show when that is somewhere.

Hsu: Okay. I want to go back to talking about a language feature, Protocols. When did that come about and how did that come about?

Naroff: That came about to support a better IPC mechanism that was language based. You know, I think Bertrand wanted better integration with Workspace and wanted other plugins and other apps to be able to message it and that's what I recall the start was. So we had a feature called-- we had forwarding, which people were using for doing prototypes of that. You could have, like, a remote object proxy, which was a forwarding object, and then forward it on, and so the interesting thing, again, it gets back to dynamic, if you give people some of these hooks you'll find some of these developers know you're busy, they have a need, so they find your hook to implement. So I think my recollection was that-- is that Bertrand did an early version of a proxy object that did message forwarding.

Hsu: And this forwarding was a runtime feature, right?

Naroff: Exactly.

Hsu: Not a compiler feature.

Naroff: Not a language feature.

Hsu: Right, right, not a language feature.

Naroff: So then-- and there will always be, even when you have compiler support, there will always be runtime support by its very nature, right, because it's packaging up arguments and stuff. So he did that

prototype and then we talked about language support and I forget. Blaine Garst hired on around that time and Blaine was in the OS group but had interest in this type, because let's face it, IPC [interprocess communication] is usually part of the OS group, APIs. So it was an example where Blaine, Bertrand and I all collaborated. In other words, Blaine didn't hack the compiler. You know, he, you know, depended on me to do that. Blaine drew up the runtime requirements from a performance perspective, from an OS viewpoint, and Bertrand was giving us the app perspective. So again, this beautiful sort of collaboration that happened more than once, certainly at NeXT, but when it comes to protocols, you know, I had then drawn up a proposal which was very simple. You know, protocols are nothing more than a set of methods that have a name and represent some abstract set of behaviors that an object's going to implement. So subclassing at the time was starting to become passé. In other words, it started to become more known that having ornate, deep hierarchies really wasn't leading to the best result, but we found that if you can just describe what your requirements were in a protocol and then your class object implements that protocol, it's really nice, right? It's liberating, because then the code isn't dependent on the class or the subclass, it's just very abstract or provides an abstraction that people liked. So I think we also added qualifiers for identifying whether the parameter was an in, an out or an in-out. There were some declarative compiler keywords that expressed intent, right. So we also did that, and I don't know whether that was the first version or second, but in general, that's something C lacked. Like, when-- even if you pass a pointer, that pointer is passed by value, right. C++ has reference arguments where the reference is taken sort of automatically. But getting back to MAINSAIL, they had uses, produces, modifies, so you knew what the programmers' intent was. So we went a little bit more in that direction because the runtime benefited from knowing that, so-- and the metadata was added. The one hiccup I recall with protocols is with a class, you had a clear home for the class. Like, when you had a class implementation there was a very obvious place to generate the metadata with the class. With protocols, we wanted metadata, but there is no good place to generate it. So I believe we used to generate them in all of the modules at first, if you used them, and then the linker would unique them. I then recall maybe later on we added a directive where you had to declare the protocols you used, and I don't know whether that was a requirement or not. I just-- it's so-- it's back so far, I don't know. But you could understand the problem, which is it's abstract, it's in a header file, and we don't have metadata other than the precompiled headers, which aren't runtime savvy, more compile time and browser time. But you could imagine moving to a model where the runtime did access header files, but we weren't set up for that. One of the things Bertrand did, now that I think of it, that-- I don't know whether he solely did this, but he, I think, he led it, was the substructure in the UNIX file system to represent a framework. Like we were using Mach-O files to cram a lot of stuff into, and--

Hsu: And Mach-O is the executable format or...

Naroff: Exactly.

Hsu: Okay.

Naroff: It's the executable format. It contains the data and the code, but also contains the metadata, and we were also stuffing icons and other stuff in there.

Hsu: Oh, wow.

Naroff: Yeah. Lots of stuff, and it was sort of like a poor man's file system. <laughs> With a--

Hsu: Kind of like the resource work on the Mac.

Naroff: Exactly.

Hsu: <laughs>

Naroff: Exactly. Same idea, and Bertrand came up with the bright idea of, "Let's stop shoehorning all this stuff into the executable file format," which I loved because I really didn't want to be in the file system business. <laughs> So he developed, like, the .framework directory structure with-- that had well-known directories in it that were the equivalent of a Mach section and would contain stuff that had nothing to do with the code or data like icons or like some other stuff, and that was-- that was really beautiful. I forget when that was done, and to this day that's a big part of what's on, you know, Apple's operating system.

Hsu: Going back to Protocols, where did the idea come from? I think one of the documents mentioned like mix-ins from-- that was in some other lang-- was that an influence? What were the influences, intellectual influences for Protocols?

Naroff: Wow. I don't-- I mean, I'd like to say that some of it or like to think that some of it might've come from MAINSAIL, but I don't think so. I mean, MAINSAIL had—MAINSAIL was weird in that it had classes that only contained data and you could have prefix classes and inherit the data but you couldn't inherit methods. So I don't think it was MAINSAIL. I don't really-- did I say in the paper at all?

Hsu: I don't remember. I think Blaine mentions in his interview something about coming from Modula, from his perspective, but I don't know.

Naroff: It would be interesting for us to pin that down, because I, yeah, I don't-- I don't remember. It's so simple that I don't even know if it has to-- well, okay. So I do remember in C++ they had a notion of a pure virtual class, a class that was purely there to express interface, if I recall. So I actually, from my perspective, wouldn't be surprised in that- that syntax they had and at least in the day, it was sort of a feature which wasn't commonly used but I thought was sort of cool in C++. So, you know, there's part of me that thinks there was some inspiration from C++ there, but I'd have to--

Hsu: Right.

Naroff: --look back in time.

Hsu: I think one of the documents that you sent me, protocols is mentioned in a section where you talk about multiple inheritance. <show document in video>

Naroff: Exactly. So now it's coming back. So I think we were always being asked for multiple inheritance in the large, and I never wanted to add it because, again, it, for a dynamic runtime, it's particularly tough and I think when I was looking at C++ I noticed you could multiply inherit a pure virtual class and a light went on saying, "Wow. Here they're using multiple inheritance just in an abstract way." That just seemed cool, so I don't know how many people were designing their class libraries with it, but I think a light went on for me, so I am-- now that it's <laughs> paging back, I do think pure virtual classes in C++ were-- because they had supported multiple inheritance, and we were being asked for it, so it was one of those things where I felt like I could sell it, A, as a way to-- and I actually now remember a presentation I did that is in the papers I uploaded, where I was selling it as certainly a solution to some of our RPC and remote message solutions but that it actively supported the multiple inheritance of interface as opposed to implementation. So while people might not have the full multiple inheritance they're accustomed to, they do now have a linguistic way to declare it, at least, and then they can implement it after the fact, so-- and it seemed to strike a chord that it was a compromise that people could live with. Yeah. So I think in-- I think Java was also influenced by that later on with Interfaces. I think they call them Interfaces; am I right? Yeah.

Hsu: Yeah, they call them Interfaces, yeah. Also the original version of Protocols only supported required methods in the protocol or what they called "formal protocols."

Naroff: Right.

Hsu: And not later on, there was this notion of "informal protocols," which was really a category on the NSObject.

Naroff: Yes, yes.

Hsu: Was kind of a hack. <laughs>

Naroff: Yeah. You know, I felt-- I guess this is one area [where] maybe I sympathized with C++ in that I felt like if you're adopting a protocol and you're a class, and you really don't feel like implementing the two or three methods or however many methods it's describing, you should be required to provide a stub. I didn't think that was objectionable. Ali Ozer and his group didn't agree.

<laughter>

Naroff: That's my recollection.

Hsu: Okay. <laughs>

Naroff: And so they encouraged us strongly to do something where it could be more informal. Yeah. Yeah. And they were our clients, so hey, fine. That's fine.

Hsu: <laughs>

Naroff: That's part of the beauty. We weren't religious. You know, there-- and that's why I think in the talk or the interview you did with Brad, there was some assumption that, "Oh, you hired Steve Naroff." "Yes, yes." "I like Steve Naroff, and he-- as a language designer," and I think the word "language designer" came up, somehow, and I'm not a language designer.

<laughter>

Naroff: You know, I'm someone who has used a lot of tools, used a lot of languages, but saying any implication that I'm a language designer is overstating the case, you know. I like to think I have good taste on some of this stuff and that I did some of the right things, but, you know, Bjarne's a language designer. You know, and again, he's been criticized plenty for C++.

<laughter>

Naroff: So just because you're a language designer doesn't mean people aren't going to have opinions.

Hsu: Right. <laughs> Let's take a little step back and talk about the culture of NeXT in the early years. What was that like?

Naroff: In the early years? And I'd say through the entire time was pretty consistent. You know, one of the things I have here, which is kind of interesting is Steve handed out, when we first, you know, like, the first year I had gotten there, it was something they handed out. "Mission," you know, "Build computers to change the world, and our friends can afford." Was kind of interesting because most of my friends couldn't afford the original NeXTcube.

<laughter>

Naroff: But it was a good goal. Good mission.

Hsu: It was an aspiration that was not met. <laughs>

Naroff: Right, aspirational. "Lead the industry in vision and innovation." Great. "Build a company so exciting and fulfilling that we can't wait to get to work in the morning. Treat our customers so well that they love our products and company, and communicate openly, honestly and respectfully," and one of the things I highlighted here, "Facts are friendly. Fear of shooting the messenger zaps our ability to perform and improve. We must continuously strengthen our commitment to management by fact and respect for each individual and encourage the free flow of information up, down and across the company," and lastly, "Raise the bar." But, you know, it's interesting, and I don't know if Steve wrote all that. Chances are he didn't, but I know he believed in it, and blessed it, and I really think the free flow of information, the "don't shoot the messenger," that was really true for the, all the years I was there, from '88 to '94, roughly. So for six years I was fortunate to work in an environment that was free of the usual B.S. that you have to deal with in companies where there're weird politics, weird turf wars. We had none of that. That I'm conscious of. I'm sure there are other people in the company that might feel differently, but I think most

people considered it to be an amazing culture where people were allowed to just do the right thing. Now, the open salary and stock thing that I briefly mentioned, that ended after I think the company grew to two hundred, which was probably after-- probably around 1990, that was discontinued. I started in '88. Maybe a little bit before '90, it was discontinued just because when you grow to two hundred, I think it was still the right idea, but it's harder to manage. The fact that it lasted for two years I think was still commendable to run a company like that. But I-- other things about the culture, Steve-- in "The Lost Interview," which I'm sure you've seen or are familiar with, it's a great interview with [Robert] Cringely. Steve talks about his early days of going to HP with the donut cart and how they took care of their people with things like that. We had the juice bar there. All that was new to me. Silicon Valley now has free tons of stuff, massage, and all kinds of stuff people get now. But in those days, just giving people free juice or-- was like a big thing. And we had cooks that would come in and cook for us. And we had actually an offsite that happened about five-- four or five months after I started. Right after the intro, Steve took us to a retreat in the Santa Cruz mountains.

Hsu: After the Cube shipped?

Naroff: Yeah, and it was at a place called Chaminade. And Bertrand and I actually shared a room. We were all-- and it was very cool. And I actually found the agenda from it and uploaded it. And it was very cool to remind myself that each day there was, in the evening, time for a jam session where people who played instruments could play because Bud Tribble was actually a great pianist. I think I broke out my sax briefly. And actually, when I first got to NeXT in Deer Creek, there was a music room for the musicians. And there were some really crazy looking speakers. And I asked about them. Oh, and there was a Bosendorfer piano, just a beautiful piano. And someone just said casually, "Oh yeah, those are the Grateful Dead's. I guess Steve got them from the Dead." <laughs> I'm like this is just too cool, right? To have a music room with the Dead's-- some of the Dead's audio equipment and piano, and a piano. I don't know if-- so, that was fun. It was just a fun group of people. And Chaminade, too, the other thing that was a little bit mind blowing for me at the time is we had these dinners with unlimited lobster tails. And that just seemed-- again, I grew up pretty modestly. But to be able to have as many lobster tails as you wanted, and it was really good lobster. So, that was also a little bit extreme. But one of the other things, on a slightly, I don't know, less upbeat tone, but to give you some visibility into Steve is during the first Chaminade, sort of where Steve was giving his first pitch the first day, one of the females, and I forget her name or don't even remember who it is actually, was taking notes on her computer. And Steve asked her to close the computer and said very sternly, "If you're not smart enough to remember what I'm talking about, then you have no business being here."

Hsu: Wow.

Naroff: Yeah so, you know, it set a tone. And as harsh as something like that is-- and I've had him say, which we'll probably get to later, some stuff to me that was of that ilk. Even when-- you know what I've learned? I learned that when he sent that type of message, he actually was usually right. It's just embarrassing to have it be said as directly and as bluntly as he said it. So, it was shocking when that happened. And I'm sure the person who it was said to-- and I don't even remember whether he actually said it directly to her and was trying to really attack her or said it a little bit more generally, and she got the

idea. I don't even remember those details. And to some degree, they're not important. But it was part of his management style, be direct. And yeah, so--

Hsu: What was diversity like at NeXT?

Naroff: It wasn't very d-- it wasn't-- in engineering, it wasn't a very diverse-- yeah, place. But that was partly because, in those days, it was hard enough to find people that did what we did. And for instance, if I wanted to hire a female compiler person, good luck. They just didn't exist. So, I think that's changing. I imagine it's changing where more females are going into computer science. I don't know what the data is. But I think, to some degree, it wasn't great. And it was-- and I'm not trying to let NeXT off the hook. I think all companies need to take some responsibility for it. But I do think that-- I know me, personally, if I had a resume of a female, it was treated equally. And it's just I didn't see many resumes in the tools area from females. I know there are some people, "Oh, that's such BS." Again, you have to have the data. You have to be armed with the data to really argue with me on that. I know it's totally unrelated, but I go to jazz concerts a lot. It's eighty to ninety percent male. Why is it? I don't know. Maybe females don't like listening to instrumental jazz. [Maybe] they like vocals. I don't know. But some things-- or pinball, pinball's very male dominated too still and a hobby of mine. So, there are certain areas where it's not that there isn't a place for females, but they aren't gravitating there often.

Hsu: Is there a cultural reason? Are there some-- some of the explanations for these are that some cultures are male dominated or are maybe hostile to women entering. You mentioned jazz. You mentioned-- these-- it could--

Naroff: I know jazz isn't hostile to women. That's why I like to bring it up because it's benign. It's entertainment. And it's interesting. The clubs I go to in L.A., the females that usually do show up are of Asian descent. And a lot of times, that's because Asia is just more plugged into jazz than this country is at the moment. So, sometimes, it's the culture these people are coming from. So, as for why our culture-- at the time I was working at NeXT in the late '80s, early '90s, why females weren't gravitating towards that, I don't have any great insight into-- there's no doubt that there's a cultural personality-- there's a personality type that's going to do well in a Steve Jobs culture. And just like what I just said, you have to have some thick skin to deal with some of the antics. Just like I, to pass the interview, had to defend myself. There's an element of that. And maybe some females aren't as comfortable dealing in that. But what I'm saying is many females didn't get up to bat. We didn't interview that many females in those days.

Hsu: But do you think that the culture of engineering itself was that conducive to women coming in, or were there a lot of women-- of the women that were in engineering, did they feel comfortable? Or did they feel like the--

Naroff: I think so.

Hsu: Or did they fit into the--

Naroff: I think so. We had someone named Kate-- oh, I forget her name. She was working in the Digital Librarian group. Oh man, but-- and I think I ran into her at one of the NeXT gatherings. I believe she felt NeXT was as great a place as any of the males there. And Cathy Novak was a documentation person. I worked closely with Cathy. She loved it there. So, I do feel the females that were there did really enjoy the culture as well. I know they're-- just like the males, you've heard of the Steve Jobs hero/shithead rollercoaster. Well, if the males are on that rollercoaster, chances are the females are. And whether they react the same way as males do, I don't know. I guess it depends on the person. I can have a thin skin sometimes. Actually, an email I wanted to read you before that Steve wrote me, I was disappointed that when Steve demoed the Lotus software, which depended on Objective-C++, all that stuff, I was disappointed that he didn't give my team credit because we did it in our spare time. We finished it. They loved it. And I had gotten kudos from Lotus. So, I just wanted our leader, in the public setting when he was demoing [Lotus] Improv, this was just internal to the company, to give us a shout out. But he didn't. And so, he wrote-- and I told him in an email that I was unhappy with that. I won't read you my email because there's really-- it's not that exciting. But his says, "I think you are being a little thin skinned here. The reason you, we, are doing C++ is to ensure the success of our computer and therefore the success of the main mission of your group, Objective-C. Going to the restroom also helps the success of your group, but we don't recognize you for that either. I think that you, as the group leader, need to keep the vision of Objective-C alive within your group and NeXT and treat C++ as a necessary task to ensure that our main vision thrives." So, there's an example where I know he loved the work that was done. But because I'm holding him to a standard, he's unhappy with me holding him to that standard. But Steve Jobs was the type of guy that, even though he's giving it to me a little bit there, he fundamentally knows leaders have to have empathy for their team. And believe me, he respected me more. So, I was fine taking the shot because I know that he stored this away as "he was wrong."

Hsu: Hmm, I see.

Naroff: In other words, he defended himself by telling me, "Well, every time you go to the bathroom, I don't thank you." But the reason he did that is he felt guilty. That's his way--

Hsu: Interesting.

Naroff: Of mea culpa.

Hsu: I've never heard that before.

Naroff: That's why I told the story.

<laughter>

Naroff: And it's funny. I've saved very few emails. But I saved that one because it says more about him than it did me.

Hsu: Hmm, interesting.

Naroff: Yet-- again, he's trying to make me feel bad. "You're thin skinned. God, you go to the bathroom, you know, what's up?" And I don't think I was being thin skinned. And I don't think he really thought I was being thin skinned. I think he felt bad. But that's the type of relationship that was complex to understand sometimes. And some people, after getting his response, I'd say probably a lot of people would say, "God, I shouldn't have sent that. I don't want Steve to think I'm thin skinned. I want him to think I'm a tough guy." No, I was totally behind that email and had no problem with his response.

Hsu: So, you owned it.

Naroff: Totally owned it. There was no reason not to own it. There-- listen, there are other things. I played my Steve Jobs email cards very carefully. There are definitely-- you could have-- I could have imagined sending him emails where he would have been totally right. Man, I'm being thin skinned. I'm a jerk. Yeah, I'm expecting too much from you. But not this one. This one enabled major third-party vendors. This had an impact on the sales of our computer. And how often do people who work on plumbing and compilers have an effect like that? And all you did was have to thank the four-person team because there were debugger-- I mean it wasn't just me. I wanted it for the team. I had already been thanked. So, it wasn't about me. And he understands leadership. And he understands selflessness and again empathy. And I think he was just unwilling to admit I was right. He didn't feel-- he wasn't in the mood to say, "You're right. I'm sorry."

Hsu: Interesting. Could you talk about what it was like when NeXT had to stop making hardware? How difficult was that?

Naroff: It was very difficult. Yeah, I felt like so much of the sizzle with NeXT was surrounding the industrial design, the manufacturing, and just the whole shtick about developing computers for our friends, having a workstation, the power of a Sun with the UI of a Mac. And it never became affordable for our friends. And I think Steve's heart was in the right place. I think the company's heart was in the right place. And we just didn't succeed at that. Obviously, we had to get out of the business. So, it was bad. I guess NeXT got out in '92, is that when it was?

Hsu: I think some-- yeah.

Naroff: Late '92.

Hsu: Something like that, yeah.

Naroff: So, I was, as you can imagine, fairly tired after all this bootstrapping of NeXT and all this work. And we, as I discussed before, had a failed pregnancy in '88. And my wife-- we wanted to try again. And we actually had our first child in '94. But in '92, we were starting to think about it, starting a family. And I, just like the moment when Cericor was bought by HP, and I started thinking about other opportunities, when NeXT got out of the hardware business, it was kind of similar. For me, it was wow, I've put in some tough four years. It was '92. And I need to start thinking about the future because I don't see NeXT making it as a software only company. And one of the things we did around that timeframe, Steve

organized a trip to talk to Bill Gates and his guys up at Microsoft because we were now on Microsoft's turf.

Hsu: Right.

Naroff: Right? And so, we went up to try and sell them on modifying Visual Studio to work with Objective-C better. We were also selling them on some other stuff. And I don't really recall all the details. But it was a contingent of Paul Hegarty, who was running software at the time, myself, I believe Avie was there, and Steve. So, we went up to Microsoft, met with them. And it was an okay meeting. To be honest, I don't remember much about the meeting. I remember more about an interesting-- we got to the airport, and we rented a minivan. And it was a Ford Windstar. And Steve said, "I'm driving." And Steve's driving this Ford Windstar. And he's bitching and moaning about it. I'm like, "Steve, it's not a Mercedes, okay? It's not a sportscar. Calm down." <laughs> And he said something like, "What kind of idiot would ever buy a car like this?" And my wife and I had just bought a Ford Windstar.

<laughter>

Naroff: And I didn't tell Steve that. That's one thing I didn't disclose, right? And so, I remember-- but again, he's-- it's just so funny. He's treating this thing like a sportscar because, you know, all he ever owned was a M-- a gray M-- a silver Mercedes, sports coupes, or something. So, it was a fun trip. But it was more evidence that man, we're on their turf now, and it's going to be tough. I mean Win32 programming was just an abomination. God, it was just awful. I can't imagine programming in that environment. But all their people were accustomed to it. So, I know we had better tools. I know it would have been great to work with them. But they were the ten-thousand-pound gorilla, and we were just a small software shop at the time. And believe me, they didn't want to become dependent on us. So, I think they might have thrown us a couple bones, but there was very little synergy, real synergy, with them. Yeah, so the companies I interviewed at around that time period was First Person that was doing Java, a spinoff of Sun, and actually Apple, the research group in Massachusetts.

Hsu: Oh right, the ATG, they were doing Dylan.

Naroff: They were doing Dylan.

Hsu: Right, another dynamic language.

Naroff: Exactly. And since I had had some visibility in the community for being the person behind some of this Objective-C stuff and NeXT technology, I had lots of opportunities. And several principal NeXT people went over to First Person. Mike DeMoney, the guy who originally hired me, went there. And him and Peter King were there, I believe. And they were talking me up. And I had a great interview with Gosling. He's such a nice guy. And he's such a bright guy and a humble guy. I really wanted to work there with him on one level. On the other level, I didn't want to just follow my NeXT colleagues. And I really didn't feel Sun was totally behind it. I felt like they spun First Person off, and they were-- it wasn't-- it wasn't firing on all cylinders because the orig-- you may remember, First Person was tasked with making

software for like almost embedded devices. The web browser angle was nowhere to be found at that time. Right? So, I was skeptical that was going to be successful, though it looked like a cool language. And they were very loving of Objective-C. And so, it felt like an interesting progression, but I didn't end up-- I was offered a position and didn't take it and decided to hang at NeXT a little bit longer and also did the interview with Ike Nassi who was running a chunk of Apple Software for a while. And they too had amazing people, Mike Kahl, who was the author of Lightspeed C, which was a great product. And he's just a great guy, David Moon, who was one of the fathers of garbage collection, Andrew Shalit was there. They had a wonderful team. They were a research group. And their product did not hit the milestones necessary for Newton. And then they had no clients. And they were looking to bring me in to help solve that. It was mission impossible. And so, I pretty much thought that they were going to close that shop down. Like if I moved to Boston, I was convinced a year later, I'd be moving to Cupertino. And that happened. So, I held tight and then became pretty interested in Taligent. And rather than work at Taligent, I decided to work at the mothership. So, I took a job with IBM in North Carolina where a lot of the leaders and management are located. The attraction was get out of the rat race of Silicon Valley. The cost of living there was much improved. We had a very modest place, just to give you an idea, a modest place in Fremont next to a train. And for that same-- the same price of that very modest house got a beautiful house on a golf course in North Carolina. And I took a job as an IBMer. And when I left, Steve said, "You're not going to be happy. I know you. It's not the right place for you, but go do what you've got to do. It's probably a sabbatical." And he ended up being right. I lasted nine months there.

Hsu: So, you were working on, what, VisualAge C++ there?

Naroff: I was helping-- I was helping their management, upper management, understand whether VisualAge and other initiatives, OpenDoc, Taligent, whether these efforts were going to bear fruit.

Hsu: So, you were almost in a consulting role. Hmm.

Naroff: Yeah, I was working for a guy named Rod Smith who was just a wonderful-- he's a fellow at IBM. I love Rod, great guy. I actually just-- all the people at IBM, just good high integrity people. So, it had nothing to do with the people or the cul-- well, the cultural aspect was I wasn't really comfortable being an outbound mole who shook hands a lot. I mean really what they wanted me to do was get in the trenches and find out stuff that, in a sane world, they would be figuring out, which is, are-- so one basic thing is well, aren't you guys using Taligent? You want to ship it a year from now, but no one's running it.

Hsu: <laughs>

Naroff: So, there were so many basic aspects of software engineering that they were missing, eating your own dog food, using your own software. Why would you hire someone to tell you whether something is useful? But it's big company stuff. Right? So, Taligent they defunded. OS/2 they defunded. OpenDoc went away. So many things went away. That happened after I left. And I'm not trying to say in any way I was responsible for decommissioning any of that. That's part of why I left is because one of the things I don't like about management is firing people or canceling projects. It's not fun.

Hsu: What was it about Taligent that initially drew you to it. And how do you compare it to NeXTSTEP?

Naroff: I can't compare it to NeXTSTEP. I know that they were, at some level, very fond of NeXTSTEP, but they tried hard not to copy it. And let's face it. They were in C++. So, at least from a linguistic perspective, we knew they weren't going to copy it. But the thing-- they were naïve about building a system of the scope they were building in C++. They got bitten by the same bug that bit HP with Cericor, same exact bug even though it was years later. C++, they realized they had to actually control the compiler and the code gen to evolve it with their system like we were doing with Objective-C. But C++ was a much bigger bear to wrestle than Objective-C. One of the-- I mean we didn't talk about this before, but an important point that I always love to make is keeping the language simple has two values. One is people looking to use your libraries only have to learn a small set of programming idioms. Number two is the programming team, when you have a problem, the surface area of what can go wrong is far less than with C++. C++ is such a complex language that, if unbounded in what people are allowed to use, the nooks and crannies can drive people crazy, <laughs> especially compiler writers. And so, they were stumbling on performance, very interesting, right? People were dinging us for, "Oh, it's dynamic. It can't perform." They realized that when virtual tables get too big and that you have lots and lots of composite subclassing with many levels, that the implementation wasn't up to what they had imagined. They're very bright people. They did a lot of good work. But aside from infrastructure problems, they were also just not eating their own dog food. I mean they had IBM *and* Apple directing them. And when you have two big companies that are directing something but neither company is running that software on a daily basis, it's a joke. It's-- so, I felt like okay, I'm at IBM. It's so easy to see the problems. Why am I even being paid?

<laughter>

Naroff: If I were a different type of person and was really ready for retirement, I could have said, "Let me just kick back, shake hands." I just wasn't-- I was so accustomed to building stuff and feeling good about it that just playing this weird sort of poli-- techno-politics was-- Steve knew I was just going to-- plus, so many of these people were running green screen mainframe-based email. They were-- we would do our-- I went from NeXT rich text, very inter-- Steve called it interpersonal computing, sort of the precursor to what we have today with the web, to going to IBM and having to send emails in green screen sort of text only. And it drove me crazy. In fact, I have a couple emails that I saved from when I was leaving from IBMers. And it's in that mainframe format. And so, it's tough to take a company with their background and culture and modernize. So, yeah, they tried to modernize by funding Taligent. And it just wasn't happening. But to get back to part of your question, there was a guy named Erich Gamma who was there, who was one of their principal architects who I had immense respect for.

Hsu: He's one of the gang of four, right, who wrote the *Design Patterns* book?

Naroff: Got it. Yeah. Yeah, I was a big fan of that book and thought the world of Erich, and I believe tried to hire him at NeXT. And Erich-- I just thought Erich was going to figure out, along with other Erichs at Taligent, how to make it successful. I didn't realize until I was on the inside how off the tracks it really was, though technically, their hearts were all in the right place. And I think there was a lot of over engineering going on. And when you don't have people using your software, you are evolving it in arbitrary technical

directions that may or may not become fruitful. So, they just weren't driving it in a sensible fashion. And that's getting back to culture. It wasn't about technical prowess. Taligent had-- I'm convinced Taligent had as many smart people as NeXT did. And I bet some could argue smarter in many regards. But one of the brilliant quotes from Alan Kay in one of the things I read recently was, "Perspective is equivalent to 80 IQ points," right?

Hsu: Or point of view, I think.

Naroff: Point of view, there we go. Okay. Similar, perspective, point of view, but okay, point of view. And I think NeXT had that in spades. And it had it because it was "the sum is greater than the parts" type of thing and the collaborative nature of the culture and the fact that everyone, Steve included, was always using stuff. Like actually, one of the tools my group did that a guy named Mark Wagner, who's now at Microsoft, when I left, Mark took over the team. One of our-- we used to have weekend hack. I forget what they called them, but-- or maybe it was a day thing. I don't know if it was a one-day, two-day, or three-day but basically, use the APIs, the kits, and develop an application. And Mark developed this really cool side-by-side differencing program. And one of the coolest aspects is when you scrolled, they would synchronize. And he did a wonderful job on that. And I brought Steve in. And Steve was blown away. Steve's like, "Oh my god, lawyers could use this," and he just saw so many applications other than programming tools because this was basically a better graphical diff. And that became part of actually Project Builder and the editor eventually. We integrated it, side-by-side scrolling and differencing. But there's an example where Steve, as soon as he saw that, supported it, used it, drove it. And he used everything we were doing. And that makes a huge difference. So--

Hsu: So, then you decided to leave IBM and go back to NeXT in '95?

Naroff: Yeah, Java was starting to get a lot of attention. And so, my pitch to folks at NeXT-- I forget if I called Steve or not, but I basically said, "Listen, I'm thinking of leaving IBM. I don't want to move back to the Valley because I, right now, I love--" we just had my son, Dylan-- or we were about to have my son, Dylan. I think we just had him. "And I'm happy with where my family is now. I get to see my relatives in New York and Florida. But I'm willing to work on Java remotely because I think you guys would benefit from having a Java strategy now that you're a software company." And I pitched like Java bridge stuff, just porting the VM. So, they were like yeah great because it wasn't easy to find people to port VMs and do this type of work. So, I did it remotely and would go back and forth from Redwood City, which is where I guess they were still based. Yeah, Redwood City.

Hsu: When did they move to Redwood City?

Naroff: That happened-- I think Redwood City was '90? Yeah, '90. Yeah. And I got to work on the Java language spec with Gosling a bit and my-- was listed as one of the reviewers. And it was a happy time. I got to be involved in Java. But that's how I ended up working on Java just at a different place from a different perspective and contributing a little bit. But for me porting the VM to Mach was a fun project, so I did that. And then worked on the Java bridge so that you could, from Java, interact with our kits and that kind of stuff.

Hsu: So obviously, because you had interviewed at First Person before, you had already been exposed to Java. Why did you think that Java was important to NeXT in particular, and why were you excited about Java?

Naroff: You know, I think as a hardware vendor and an OS vendor, pedaling your own language is a small incremental thing to foist upon people who are betting their business on you. Take away the hardware, take away the OS, all of a sudden the language is a bigger deal, right? It's more of an obstacle. So I actually felt strongly, even though I was Mr. Objective-C, that, as a standalone software company, Objective-C shouldn't be part of the strategy. I know that sounds radical, but I really felt that way. And it was one of those-- again, it's the real nonreligious side of me that said, listen, so many people that had so much built up in a particular technology-- so I had five or six years of my life built up. An awful lot of people would have tried really hard to ignore the business issues, but Steve really respected my honesty and that was part of why Steve and I never had big problems is because, when he asked me a question, I-- so he asked me how important I thought Java was going to be, and, from my perspective, I thought it was going to be really important. Listen, I knew that we were not going to rewrite all that code in Java. I'm not naïve. However, if the bindings were good enough, then, for all intents and purposes, people wouldn't know what's under the covers. I mean, Java supported native methods as it is, so the fact is, it's just a kit with Java, with tons of native methods implemented by Objective-C. So I just thought that was the future if NeXT was going to stick or stay as a software-only company. And it was always a balancing act, because I tried to promote Java as best I could without being anti-Objective-C, so I had to-- because you don't want to scare everyone at the company. They had already gone through some trauma with not supporting hardware. So I was enough embedded in the culture that people trusted me to do the right thing in this area. If they would have tried to-- if what I was trying to do was being done by someone that was new to the company, it probably would have been met with more resistance from Ali Ozer and the other guys. So that's why they took me back so quick. It's also interesting that, at least from my perspective, Steve Jobs was extremely loyal to me throughout. There are many CEOs that would never take you back after leaving them at such a critical point. But he was totally human about it and totally understood what I had been through and, by virtue of saying, "Well, worst case it's a sabbatical." I really didn't plan that. I'm not that type of person. It's not like I was looking at IBM as a sabbatical. Again, they're great people and it's a great company. It's just a big company that had a culture that was shocking to me. And I was very honest with the IBMers when I was leaving. I liked all of them, but they knew the culture was what it was. I mean, they had 200 people in Toronto working on compiler back ends. Two hundred. That was-- you know, I had my four to six person compiler team <laughs> that did debuggers and-- so it's a world that I didn't get. There was lots and lots of legacy. I mean, back when they were doing Fortran, and Fortran was the cash cow on their mainframes, they could afford to have 200 people working on the back end. And, let's face it, they still have cash cows, so who am I to question where they're spending their money, but the amount of money being wasted was silly. I mean, just think about the sum of OS/2, Taligent, OpenDoc. Crazy. That's a big number. Which is-- part of what I find interesting about the industry, and another area I've been so lucky, almost all the software I've worked on has been used by real people for real things. Man, that's a real blessing, for lack of a better word. Because so many people work on software that never really blossoms. I mean, OS/2 did have its day, but it sort of had a horrible death, too. And OpenDoc never really had its day. And Taligent never had its day. As you know, that's part of why Apple ended up

buying NeXT is they were unsuccessful at doing an operating system on their own and they ran into the same problem that so many big companies run into. "Oh, well, this is such a hard problem. Why don't we partner with another really big company so it can be an even bigger problem." Really? I mean...

<laughter>

Naroff: But so many people are driven by building empires. So.

Hsu: I want to talk a little bit more about the Java bridge and that work. I think one of the documents you sent was saying that like the Java bridge was a separate strategy than the Java VM. They were like competing strategies or something, is that correct? Or were they complementary?

Naroff: Complementary.

Hsu: Complementary, okay. But the bridge solution is a conservative solution, right? It's not like-- you touched on that. It's not like you're ripping out Objective-C. It's just sort of a way for Java to interact with the existing Objective-C frameworks.

Naroff: Right. It's a way to allow Java-- native sort of pure Java that has no relation to the AppKit to coexist with Java wrappers that plug into the AppKit. So in a way it's a little bit similar to the Objective-C++ solution in that what it isn't is-- like for example, it supports composite object work. Where you have a Java object that's written, again, in pure Java, no native code, and it's calling out to, let's say, UI elements that are native Objective-C. That's different from sub-classing, right? It just has a handle to that. The object models are more integrated, certainly, than C++ because Java-- under the covers, the Java wrappers, even though the bridge has some magic in it, the Java wrapper-- Java had the native method facility for us to plug into, right? Because Java, being a distinct language from C, they knew they had to plug in to C code, which-- that's what the native support is. So we're just leveraging that. There's no way in those days or even today-- I mean, let's face it, if you're a new language, you do need a way to call C usually, if your operating system's written in C. So that's the level of integration as opposed to a totally unified model. Obviously it's not like their-- like their Swing and our AppKit are just totally distinct. You wouldn't be taking a Swing object and having tight integration. That would be totally separate. Either you're doing your UI in Swing or you're doing it in Cocoa, but you'd use your higher level or lower level objects to work with either of those libraries.

Hsu: Right, yeah. Was it a good match? Java-- there are some similarities between the two languages. You mentioned that a lot of the NeXT people had gone over to First Person.

Naroff: Right.

Hsu: The Interfaces feature is very similar to the Protocols feature. I guess there is a dynamic runtime, but Java is also a statically typed language, I guess?

Naroff: Java's more statically typed. Java had no notion of "id." Like, as you know, in Objective-C, the only data type it originally supported was "id." Then we added static typing. Still dynamic binding, but static typing. And Java pretty much only has static typing. There's no way to say, "Give me a reference to any object in any hierarchy." But if I recall-- because, again, I've been out of it for a while, Java was single inheritance, right? And also with a single sort of omniscient root object. I don't think it was hard-wired into the language, but Java.lang.Object was equivalent to our NSObject. So even at that architectural level, they were pretty similar. So the good news there is people learning Java, when they learn our kits they're going to see some similarities, right? So that's the win there, whereas with C++, if someone vends-- C++ has so many features. And if you're using it in an unbounded fashion, that would really be a wrench in the works for-- if you're coming from another language. So I think just-- there was a good match between Objective-C and Java just at a teaching level, which means a lot. At the time, Java had-- Java's implementation is more dynamic than Objective-C, but Objective-C, by some definition, is a more dynamic language even though it's more compiled. Java was totally interpreted. In fact, one of my problems with Sun and Java, which we may talk about as we move on in the talk is-- later on when Apple was doing stuff with Java and I wanted more of a native compiled Java for the operating system, Sun wasn't really a fan of that.

Hsu: All right. So we were talking about Java at NeXT. So at this point NeXT is working on WebObjects, right? And that's their major push?

Naroff: Yes.

Hsu: And so you're pushing for WebObjects to be-- there be a Java interface to WebObjects.

Naroff: Absolutely.

Hsu: How far did that get while you were at NeXT? Was that actually done while you were at NeXT?

Naroff: Yes. Yes, it was done and I think it shipped. Pretty sure it shipped.

Hsu: How well was the Java work received at NeXT? It seems like-- well, I guess the WebObjects version shipped, so it seems like it was fairly well-received?

Naroff: Well, I think the people that would probably be more vocal is the people who are using WebObjects, right? And I don't have vivid memories of that. I believe it all went well. Yes. I think at the time they were pretty much pitching programming in Java when you're using WebObjects.

Hsu: So a lot of that came from the users, that they wanted to use Java.

Naroff: Yes, it's an easier sales pitch to say, "Use our WebObjects infrastructure and you can do it from the language you're currently learning, whether it's in a university or elsewhere," because Java was starting to really catch fire. A lot of people were learning Java and, at the time, as you know-- I forget exactly when Swing was released, but even AWT in the browser, as lackluster as that AWT was, it was

still exciting people because it was, “Wow, I can run my code in the browser,” right? So there was a lot of excitement. And Java was getting a lot of mindshare. A lot of mindshare. Do you remember?

Hsu: Yes. <laughs> I was in college at that point so I remember that transition.

Naroff: You know, even Doug Lea at Oswego, he was writing books on C++. He started retargeting a lot of his books and thinking to Java, so even at little old Oswego they were globbing on to Java, too. I don’t know how much it’s taught now or how much it was ever taught, but it was really popular, and I think we’ve talked offline-- I think this is probably the time period-- maybe it was before. I don’t know what the time period was, but I had said to Steve, “Is Objective-C part of our problem in terms of adoption?” API adoption, whatever. And he said, “No, I don’t think so. Our problem is we’re not selling enough computers,” so I guess it was right before we got out of the hardware business. But even later on at Apple, Steve was still a believer in Objective-C and he just felt, as a leader, that it’s his problem to sell more machines, devices, and when that happens, Objective-C will be used like we would like. The programming languages are much more about reach of developers, which makes total sense. It makes total sense. And it describes why most programming languages don’t make it, because it usually has to be bundled with a bigger revolution, so to speak. So Java’s revolution was the browser. And, as you know, things changed and Java had an interesting journey, some of which we’ll get into, I guess, but yes. Everyone thought it was going to take over the world and everything-- applets would take over the world, and that didn’t happen.

Hsu: Well, let’s get to the Apple transition then. What was that like, when you found out that Apple was acquiring NeXT?

Naroff: It’s interesting, because one of my visits when I was doing the Java work, I was upstairs where Steve’s office was, and I saw Michael Spindler in his office. And it looked like a really serious-- and this was before.

Hsu: So Spindler was still the CEO of Apple at that point? So that was pre-Gil Amelio?

Naroff: Yes. And it looked like a really serious discussion. Spindler looked all stressed out. And I just remem-- I just noticed it and then sort of put it in the back of my mind. I actually, at that time, didn’t realize that there were very serious talks going on, but-- then put two and two together. But when it actually got released it was wonderful news. It was an absolute perfect ending to what we had done at NeXT because we knew that the software was good stuff, and it just needed a hardware platform to be successful. And Apple was absolutely fledgling. None of us were, I think, bold enough to think that we were going to solve all of Apple’s problems. I think just the opposite. I think I probably thought, “Wow, that’s great, but wow, this is going to be an interesting ride.” Let’s see. What else about that time period? I was still working remotely. I was still in North Carolina, and I think Steve had called the house, and my mother-in-law picked up and said, “Steve Jobs is on the phone.” I’m like, “Steve?” He never-- he's not that type of person to call on my home phone. He said-- and I do think-- no, this was after the news. Yes. So it was after I already knew it was being sold, and he said, “Listen, we’re deciding what the management structure’s going to be.” That’s right. It was like maybe four to five months after. And he

said, “Avie and I want you to come back and run the Tools team for us.” And it was a tough one, because we really did like raising a family in a more humane cost-of-living area. The NeXT stock I had was certainly a nice bonus from the sale, but it wasn’t enough to retire on. And so I basically made a decision to go follow, I guess, my heart and just go out there again. And Steve and Avie made me a great offer to do that, and they were-- what was I going to say about Steve? They were just great about the whole transition. But I had-- oh. I know what I wanted to say. I knew nothing about their strategy about bringing the company back. And soon after I accepted and arrived in Cupertino for the first meetings, Steve started talking about the colored iMacs. It’s funny. The technologist in me said, “Oh my God. I just moved cross-country.” Again, it was a good offer, but, “I’m moving back for colored-- to make colored computers, thinking, this is going to be the solution to Apple’s problems?” And it just goes to show that, while that may have been a valid thought, that I wasn’t seeing the big picture. Because Steve was a showman. Steve was going to put a stake in the ground. Here’s some pizzazz. It was still Mac OS 8. Had nothing to do with anything at NeXT. So it was purely a hardware play, because we were trying to figure out how we were going to merge the soft-- how were we going to evolve? So NeXT stuff was really nowhere to be seen. So that’s another reason. It’s like, okay, they’re not really using our software yet. That’s going to be a hard road to hoe, and the colored computers, but he ended up being right and he knows way more than I do about what the masses are going to-- what he could sell them. He was brilliant at that. He was the best at knowing what the market wanted, and his talks were always spot-on. It’s good that I didn’t know, because, honestly, I could have done something stupid and said, “Oh, my God, that sounds stupid. I’m just staying put. I’ll go back to IBM.” Could have done that. I could have said, “You know...” I could have gone with my tail between my legs to the folks at IBM and said, “You know, I’m not so sure this NeXT acquisition is going to be right, so why don’t I continue working on whatever you guys want.” Because I knew that IBM would agree. There are so many other perks working for IBM, and it was unclear whether Apple was going to fly with NeXT. Plus it was sort of unclear what Steve’s role would be. It was just-- it was going to be more politics. Is he really going to have the freedom to do what he wants? So many unknowns. But, as you know, it all worked out in the end. There’s no doubt, though, that merging the NeXT team Tools guys with the Apple Tools guys-- part of the really difficult part of my tenure there, it was tough.

Hsu: Right. So you were brought in to manage that team?

Naroff: To merge the two teams.

Hsu: To merge the two teams.

Naroff: Yes, which was, I think-- starting out maybe like 50 to 60 people. Which, for me, was a lot of people. I mean, I wasn’t in any way intimidated by the sheer number. I was just very aware at how traumatized the Apple folks were. It was so tough going from an environment where management was respected to an environment where management was disrespected because so many people had felt burned by their years at Apple. Because Apple was dysfunctional for quite a while. So there were some damaged people. That was tough.

Hsu: Were there any other cultural differences that you needed to work out between the two sides?

Naroff: Oh tons. They-- as far as compilers go, there was quite a bit of skepticism and maybe even hate towards GCC. They had their home-brewed compiler. Mr.C was its name. And even though Apple was investing in that compiler, and it was a very good PowerPC back end, it was slow, and it was getting its butt kicked by CodeWarrior. One of the problems there was CodeWarrior owned the developers. CodeWarrior's developers loved the fast turnaround time. The code gen was good enough, and Mr.C got very little exercise other than internally. And I'm sure there were some external-- I'm no expert on that, but my intuition says, based on my pretty deep knowledge of Metrowerks, that they owned a significant chunk of the developers.

Hsu: Do you think that was because of the IDE? Because I mean, they had-- CodeWarrior was a full IDE.

Naroff: It was the IDE.

Hsu: Right, but Mr.C, you could only use that from MPW, which was pretty out of date by that point.

Naroff: Exactly. And so we were going to stop-- we did stop development on Mr.C, and I had to take those people and get them to work on GCC. And that was difficult. Very difficult. I think most of them either went away or I had to tell them to go away. I don't remember details. But I tried to mentor some folks through that and had moderate success, but not much. Not enough. But it didn't really matter. I mean, I think when you have a merge like that, some people are going to get left behind. I like to think of myself as a mentoring manager, but you have to know when to cut your losses. I can't say I was really great at it at the time, but I learned quickly. Yes. And there were always arguments about, "Okay, so we're using GCC. Well, are we going to generate function calls like the PEF object file format or Mach-O?" There was--

Hsu: Right, the classic Mac OS used PEF and NeXT used Mach-O, so there were these fights.

Naroff: Yes. So if we weren't fighting about, "Oh, Mr.C's a better optimizing compiler than GCC," there were fights about PEF being better than Mach-O. So I had to referee those fights, but Avie's take, and I reported to Avie, was, "Listen, we're doing GCC and Mach-O, so just tell them that's what we're doing." And I could understand, from his perspective, that was totally valid advice to give me, but you know when you're managing a team and you're trying to be respectful to people who are domain experts, there was definitely some entertainment of, "Okay, if you feel this is so superior, tell me why." And in so many cases, people couldn't articulate it because they weren't experts on both. <laughs> They were only experts on one. So at a very minimum I had to give them a little time to say, "Listen, study Mach-O. Come back to me and let me hear you out, because maybe some of these ideas could be worthy of adapting to GCC. I don't want to throw the baby out with the bathwater, here." But in almost all cases none of it made much sense because-- an example is: if somehow PEF saved a couple instructions on a function call, but calling a function indirectly like we do in an object-oriented setting is more common, and that's more efficient, you just have to talk about numbers. At the end of the day, you can't talk about these things sort of just willy-nilly on, "Look at these two instructions are better than these four instructions." Well, how often do we execute those instructions? There's all this analysis and no one

ever brought me amazing analysis. They would just bring me micro things of, “Look, this is--” It was so frustrating. It was so frustrating.

Hsu: Right. So like penny wise, pound foolish kind of--

Naroff: That or, again, they’re just naïve with the way the NeXT system was operating because they knew nothing about Objective-C and nothing about the way the dynamic shared libraries worked, or so on and so forth. But maybe even more importantly, which was Avie’s perspective, is: I don’t care if any of this stuff is better. Our stuff has been proven to work well, so why are we talking about this? Aren’t there more important things? So the interesting thing, when you have two cultures that are combined, if you just uniformly take Avie’s approach of, “Shut up and just do what we’re doing,” it’s tough to manage a group when you’re showing no empathy or respect for what they’ve done. So I was in this middle. I was in the middle. It was tough. It was not a fun time frame for me, because, let’s face it, as we just discussed, I had a small, finely-tuned group for most of my tenure at NeXT. I still had plenty of time to contribute good engineering work. And so now I had no time with 60 people or however many there were. I was purely [e]mail, management, reprimanding, cajoling. It wasn’t a technical job anymore, and it was frustrating. And to be honest, over time I realized it also affected my health. I just was not a happy person.

Hsu: You mentioned CodeWarrior. On your resume, one of the big things that you were doing at the time was managing the relationship with Metrowerks. How important was that?

Naroff: It was mission critical, as Steve likes to say, because-- let’s just take-- they owned our developers, right? So, for example, getting them to support and interoperate with Objective-C: critical. Getting them to support Carbon: critical. So they were always sort of trying to hold out their hand so we would put money in it to motivate them to do various things. So it was a really tricky relationship. I had a really good rela-- because their principals knew I was a compiler guy by trade at NeXT and was doing lots of the work on the compiler, because of that they respected me to do technical negotiations on a lot of this stuff. Greg Galanos, their CEO, was very technically savvy. I don’t recall whether Greg was actually a compiler guy at any point. But Greg and I had a great relationship, and I had a good relationship with Andreas Hommel, who was their lead compiler guy, and others there. We coached them on changing their compiler, but that took real work. I had to write ABI documents, get them the-- as much detail on plugging into our runtime as possible. But it was an interesting dance because, on one level, they owned the people Steve wanted me to own. What we all wanted was for Project Builder to be good enough to compete with CodeWarrior. And that wasn’t going to happen any time soon. And, again, we’ll get into that story eventually, too. But that wasn’t going to happen overnight. So I had talked with Greg, who was pushing it more for Apple to buy them. To purchase the company. I thought from the beginning that buying the whole company was unlikely to fly with Steve. But buying the parts we wanted might fly. So there was lots of trying to divide up how it could work. Emails that were written. I found some of those emails in my folders. In the end, we just kept on getting them to do what we wanted. We never bought the company. I think Motorola bought them eventually.

Hsu: I think so, yes.

Naroff: But by the time Motorola bought them, we were picking up enough steam as a company and a development tools team that it wasn't damaging. But from '97 to 2000-ish, it was painful to [be] beholden to them. In fact, it was interesting. When I was reading some of my appraisals, trying to refresh my memory on what had happened over the years, I see Avie in my appraisal dinging me for trying to push the Metrowerks acquisition, where he was saying basically, "Steve needs to mature on the business front because he's pushing unrealistic-- this acquisition which makes no sense." He's valid to his own opinion, but, interestingly, I also found an email from Steve Jobs in '92 [2002] that said, "I should have listened to you back in the day and bought Metrowerks." It was really weird how it came up. I have that email somewhere. Because we were talking about something else, and we were sort of over the hurdle, and I don't think he ever copied Avie on that, and I certainly wasn't going to show Avie. It didn't matter to me at that time. But it's one of those things where-- I loved working for Avie, and he was a great guy and a great technologist, but just as he thought I could be overly emotional about some things, sometimes I think he thought his opinion was gospel, and it was odd. He's totally entitled to his opinion that we shouldn't buy Metrowerks. That's a valid opinion. It's invalid to say my business acumen was flawed as a result. In other words, why was I even trying to have a great relationship and coddle it so that we could have purchased them. In the end, my relationship was so good with them that I could have gotten us a better deal, and divided up the technology, such that we would have just paid for the important things. But it never got to that point, because Avie was never a fan. And you know, there's no right or wrong. I just don't really appreciate being dinged for something where there's no right or wrong. It's just so, so obvious that they owned our developers, and that Apple was a big enough company to buy the developer company that owned your developers. It's just really simple. And they had a great C++ compiler. And we wanted that to do Objective-C++. I don't think they ever did that, because that was more money. But it was interesting times.

Hsu: So let's get back to Java, then. How did you-- or maybe, do we want to talk about modern Objective-C first?

Naroff: Yes.

Hsu: Which one came first?

Naroff: Well, they're related.

Hsu: Okay, okay.

Naroff: So--

Hsu: So, well, first let's explain what modern Objective-C was.

Naroff: I will. But first let me back up.

Hsu: Okay.

Naroff: So as soon as we were able to visualize Java interfaces to our APIs-- in other words, we had converted the keyword arguments to Java style naming. And even though some of them weren't pretty, we had done it. And so--

Hsu: And that had been done at NeXT already by you.

Naroff: Yes. Well, I wrote a translator to give a naïve translation, but the [App]Kit folks were free to go in and provide a more compact name. Because if you had three keywords, just concatenating them, kind of looked pretty funny.

Hsu: Okay.

Naroff: But that's [neither] here nor there. We had Java interfaces that basically looked like a standard C++ style naming. So we had done that. So I had said to myself, using DevKit, "I'm going to write a DevKit program"—which again was the infrastructure to do precompiled headers—"I'm going to write a little semantic action routine which transcribes all the Objective-C interfaces, but rather than put out a Java interface, put out an Objective-C interface, and translate the message expressions to something that looks like object-dot, just like Java, right? I think is Java dot or arrow?

Hsu: It's dot.

Naroff: Dot, okay. So I just basically said, "Let's convert the Objective-C syntax to a Java-like syntax. What that became known as within Apple was the "modern Objective-C" syntax. So I did this kind of as-- I was reading so much about how Java and C++ programmers can't stand the Objective-C Smalltalk style naming. So again, I had this tool, DevKit, revved it up, and got it to produce a transcription that, instead of Java, was this syntax that we could basically use for Objective-C proper. And I showed it to Avie, and he got really excited. And we were on the cusp of-- I mean, obviously Avie knew, I think Avie knew about the acquisition way before I did, because--

Hsu: Oh, you had done that modern Objective-C work at NeXT?

Naroff: Yes!

Hsu: Oh, it was prior to the acquisition. I didn't know that.

Naroff: I'm pretty sure I did it-- you know, it's really a shame-- it's really funny. We used to have a document, and I like-- I went out of my way to like get rid of that document. Because there was so much weird trauma related to it. I mean, I found so many documents, I couldn't find anything. And there was a fair amount written, because the Doc people even revved up to maybe make this change. I do believe it was-- if it wasn't right before the acquisition, and just related to being a software-only company, it was very soon after while we were in the chasm. Sort of in-between the time it was said that we were going to be acquired, and then the time we were acquired, which [was] many months' span. And this was a quick thing. So I may have taken a couple weeks to prototype it, and I think it was one of those things where--

Hsu: So like late December '96 to the first two months of '97?

Naroff: Yeah, I think so. But it sort of made sense as the new Apple that we might want to start fresh with this new syntax. Because it's more akin to C++ and Java. And I am-- I'm not 100 percent certain that I prototyped it in isolation and then showed it to people, or whether someone sort of said, "Why don't you prototype this?" I really don't have that vivid a memory, since it's a while ago. And the way we worked there, it wasn't even important who said, "Why don't we try this?" But so I did it, and Avie got excited. And Ali Ozer's [AppKit] team was not thrilled. So perfect example of the tension between, wow, external programmers, that have never seen Smalltalk, aren't familiar with NeXT, but just want standard dot notation. They'll like this work. But the traditional NeXT people are having a fit. The other thing to mention of why I almost felt bad that I did the work and even floated it, even though-- I mean, I shouldn't feel bad about floating it-- but why there was so much excitement being-- for something that's, to me, really, really simple and superficial. Okay? Why I ended up being against it was, again, much more pragmatic. I said, "Oh, my god! Objective-C++ has been such a big thing in NeXT world, it's going to be probably just as necessary in the Apple world." And the beauty of Objective-C++ is the Objective-C message expressions are clearly segregated and distinct from the C++ code.

Hsu: Right.

Naroff: If we make Objective-C message sends look like C++, you will not be able to tell the difference. <laughs>

Hsu: Right.

Naroff: Now, you could argue, "Oh, well, we're going to have a fancy code sense editor, and you know, the code editor will know, which is true, there's no doubt. The code editor can know, and the language knows. So there's no ambiguity in terms of the language, but just at a superficial level it makes the two languages more integrated, which is why people were excited about it on one level. So I thought keeping them semantically—syntactically and semantically distinct was better than syntactically similar, but semantically distinct.

Hsu: Right.

Naroff: And I still feel that that was the right decision, not-- so in the end, I sided with Ali's guys and said, "I can't in good faith push this." And I think Avie was very disappointed. And if my only goal in life was to make my manager happy, I would have given in. I would have-- and he would have made the resources available. But I do imagine he probably-- or Ali talked to him as well. There was enough communication still within the core group of people that Avie never-- I don't remember Avie ever telling me that Ali convinced him that it was the wrong thing to do, or I convinced him. But it's funny, because it just sort of dissipated. <laughter> It just-- and I used to have documents that-- for all this stuff. And even that, I abolished from my library. There's no remnant that I could find of any of that.

Hsu: <laughs> Oh, that's interesting. This story sounds kind of similar to a later thing that we'll talk about, maybe in our next session, but like the non-take-up of Objective-C garbage collection has-- sounds like a very similar story. <laughs>

Naroff: It's a similar-- it is a similar-- well, it's similar in the sense that you ha-- yes, you have external requirements, and needs and wants to be competitive with other languages versus the reality that our kits grew up with-- in a non-garbage collected world, and then went to a manual reference counted world, and trying to evolve to garbage collection is very tricky.

Hsu: Right.

Naroff: And yeah, so but we invested way more in that [garbage collection] than we ever did [in modern Objective-C]. We invested a lot. We were pretty tenacious on that. And--

Hsu: Right. I mean that feature actually shipped, whereas this never even got--

Naroff: It shipped, but yeah, the problem with that feature was getting apps to actually use it. It was so fancy, Hansen, that it was on a per process basis, you could select whether you were going to use automatic garbage collection, or manual reference counting. And while that's not tricky for the app, it's very tricky for the frameworks to basically change behavior. And Blaine [Garst] worked his butt off on that with the [App]Kit guys, and it took so long to get any traction. It was painful. It was painful to manage through that. And part of the pain was we converted Xcode to be totally garbage collected, but we didn't have many more mission critical apps, so it was a little concern about how-- whether the dual mode functionality was robust. We were on the bleeding edge with that. And obviously, then, you're talking about memory management and performance issues that are dependent on some weird flag, which complicates QA and everything. That was a difficult time, and that was like in '92, I believe. No, no, no. What am I talking about?

Hsu: 2000-and--

Naroff: This is 2000-- 2002, 2003. Well, maybe later than that.

Hsu: I think it was later. Yeah, 2006 or '07?

Naroff: Right, right, right.

Hsu: Right. But yeah, going back to-- let's get back to Java then. So how did you get back into doing-- working on Java at Apple?

Naroff: Well, they decided that when I merged-- when they merged the Tools teams that I should take the Java team as well.

Hsu: Oh, okay.

Naroff: At an organizational level, it made a ton of sense, since I was working on Java. But--

Hsu: So you were managing two teams.

Naroff: Well, it was actually--

Hsu: Or were they merged?

Naroff: I was-- the group was called the Java and Advanced Technologies something, or Language Technologies, or they were basically "and Java." I forget which was first. Should have my business cards here. But yeah, the team was unified. It wasn't like-- they were both under me. And you know, Java was pretty strategic then. We were investing a lot in Java because Apple-- you know, Sun didn't give us any respect at the time. They were flying high. And pretty much was un-- they were unwilling to help us or invest in it, so we were left holding the bag. But maybe more importantly is Oracle was a big strategic customer partly because of the Larry/Steve relationship, but no matter why it was, they were strategic and they were really adamant that our Java performance needed to improve. So we, actually, here's an area where I decided to go outside. I purchased the Symantec JIT, Just In Time compiler, and integrated it. They had a pretty fast Just In Time compiler. And that helped us for some amount of time. But we also beefed up our Java VM group. I don't know whether this person-- someone named Ivan Posva who was just an amazing runtime guy, I forget whether he came over from another part of Apple or we hired him, but he was working on Java as well, and was just a fabulous runtime guy.

Hsu: And so the group did the Java for both OS 9 and OS X.

Naroff: Correct.

Hsu: Mm hm.

Naroff: Yep. And the OS 9, AWT and Swing eventually, Swing work was really vast, because one of the things that we wanted to do is innovate in the area, especially with Swing, more than AWT, of having great native look and feel. You know, one of the problems people always had with Java apps is they looked pretty bad. And when you have a cross platform UI library, it's sort of common that they don't-- cross platform UI libraries don't look as good as native. So we wanted to take a crack at whether we could upgrade the look. And we did some good work there. But you know, it almost didn't matter, because Java had a reputation of having ugly UIs. Again, sort of another theme. Sun didn't ever develop the killer Swing app that people said, "Wow! Look at that! That's a great app! I want my app to look like that and behave like that." They just didn't really drive Swing from an app perspective. They didn't have that much-- I think eventually they developed a browser with it? But it wasn't a great example of a browser. So it sort of showed the limitations.

Hsu: So the classic Macintosh Java product was called MRJ, for Macintosh Runtime Java? Is that the correct--

Naroff: I think so.

Hsu: How did that differ from the OS X version?

Naroff: Well, it differed a lot, because the Java VM grew up on UNIX, since it was a Sun technology, so it really liked having a modern UNIX/Mach style operating system under the hood. And so on Mac OS 9, you know, it wasn't. So it required a lot more work. And you know, I trusted the people working on it at the time. I was not a traditional, classic Mac guy. So I don't know where all the bodies were buried on that. But fortunately had a good enough team that they kept that going and did a lot of the work. Obviously when Mac OS X finally happened, who knows when, now? But it seemed like forever. Then eventually we deprecated [MRJ] and didn't have to invest in that. And that-- all that investment in Java, it was tough to swallow, because what Steve would have loved, and I would have loved, is to invest in the tools. To really, through this whole time period, develop a killer IDE that was really going to compete with CodeWarrior.

Hsu: Hm.

Naroff: And it wasn't until I decided to step down and get involved in Xcode that we started to really invest the proper resources in it. Because it was very clear that even though we went through a time period where we thought Java was going to take over the world, you know, four years later, around 2001, after the acquisition, it was pretty clear Java was faltering on the client and the desktop. And that it wasn't the threat we thought it was going to be. And there was no reason to con[tinue]-- Mac OS X was coming online. So then we started putting more emphasis and resources behind the Xcode rebirth, which was basically a rebirth of Project Builder.

Hsu: Right, right, okay. So, but in the meantime, so it sounds like Java was essentially like a hedge, the Java strategy.

Naroff: It was a hedge.

Hsu: Right, okay.

Naroff: It was a hedge. And you know, it was important enough to some of our strategic third parties like Or-- even if Oracle was the only company that was happy we were making this investment, because it helped them, I guess you could claim that then that was a good investment, because Oracle continues to be such a major player. And at the time, you know, they were still really a major company that we-- we weren't an enterprise play, but it was clear that over time we were going to try and push more into that area. Enterprise wasn't a focus of Steve's for a long time, but and I don't really know how much his relationship with Larry influenced the investment we were making. I don't remember ever talking about that. All I know is they were put on my radar screen in terms of giving them special love and care. Which we did.

Hsu: So was Oracle using WebObjects? Why did they need Java from Apple?

Naroff: Well, just on the-- if they were running on the client, they just wanted to be as fast as possible.

Hsu: So Oracle wrote Java software that ran on the Mac.

Naroff: Exactly.

Hsu: Oh, okay.

Naroff: Or that they wanted to run on the Mac.

Hsu: Or that they wanted to run on the Mac.

Naroff: They were writing. They were all geared up. Java was going to be their cross-platform strategy.

Hsu: Oh, okay. So Java was the platform that they were going to deploy to that would allow them to run anywhere.

Naroff: Exactly. Right. So you know, if they have, as you know, they have a tremendous portion of the database market if even, you know, 20 percent of their customers want to run some of their cross-platform database apps on a Mac client, that's a big number.

Hsu: Right.

Naroff: So yeah, that's what it was for. It wasn't for bri-- you know, had nothing to do with the bridge strategy. It was much more about they just want Pure Java. Remember that term, "Pure Java" app store? <laughter> So.

Hsu: Yeah. But I mean, during this time, you know, WebObjects 5 shipped in a wholly completely Java version, no longer was Objective-C. And then I remember the Cocoa Java was being pushed as a thing. How important were these initiatives overall to the, I guess, to the company or to the strategy?

Naroff: I think most-- if I understand your question, any Objective-C API that was wrapped through Java didn't benefit as much with the Java investment we were making as the Pure Java folks. Because when you're using the wrappers, a significant percentage of your code is really in Objective-C. And some code is in Pure Java, but relatively small, as opposed to writing a Swing specific app, Java app.

Hsu: So even WebObjects was bridged.

Naroff: Exactly.

Hsu: It wasn't a Pure Java.

Naroff: I think eventually he did do a pure version. But that history I'm also-- you'd have to talk to a WebObjects person. I know it was initially bridged. And I do remember them pushing to reimplement the whole thing in Pure Java.

Hsu: Right, I think I remember that, too.

Naroff: And I do think they did that. But I don't recall--

Hsu: I think that was 5, I thought.

Naroff: Was it? Okay.

Hsu: That was what I remember, but I don't know for sure.

Naroff: Well, if they were doing a lot with Pure Java, then I'm sure they were interested in some of these basic improvements. Yeah. But I doubt they integrated with Swing, right? They were probably still-- well, I don't know.

Hsu: I don't know about that. That might--

Naroff: This is where it all gets sticky. It's like from a language--

Hsu: And it was a server side technology, so they didn't have to integrate with the client side stuff.

Naroff: Okay, they didn't. Okay, it was-- that's right. It was just server.

Hsu: Right. So it made sense for WebObjects to go pure Java because they were only on the server. Cocoa Java was a different beast, because it's a client[-side technology].

Naroff: Well, there's no doubt that we had some server clients, but that wasn't where Apple was making most of its money.

Hsu: Right.

Naroff: But you are right. Whatever few client-- enterprises we were running on Mac OS X, benefited from those improvements, yep.

Hsu: And at this point in time, who did you report to?

Naroff: Avie.

Hsu: Avie directly? Okay.

Naroff: From '97 to 2001, I'm pretty sure for that whole time I reported to Avie.

Hsu: And at some point, I think you reported to--

Naroff: Well, he left. I forget which year he left. Oh, he left way later than--

Hsu: Well, he left later, but I think he stopped being the VP of Software at some point, and became like a special advisor to Steve directly?

Naroff: Yeah.

Hsu: And then so Bertrand--

Naroff: That was after 2001, yeah.

Hsu: Right, so yeah, I think it was after OS X shipped, and then Bertrand then replaced him as the VP of Software.

Naroff: Right. No, so it was mostly Avie, I'd say.

Hsu: Right. And then there was like Ted Goldstein at some point was the head of--

Naroff: Tools.

Hsu: Tools, overall.

Naroff: I hired Ted in 2002 because I wanted to step down and get my hands dirty again with Xcode. And so I told Avie, I wanted to hire my replacement. And Avie and Steve begrudgingly let me do that.

Hsu: Right. But yeah, so talk about sort of this-- you know, you've mentioned earlier, but this sort of arc of Apple and Java, and you know, sort of the-- Java's rise and fall on the desktop, and whether or not, you know, how well Java was fully supported, and the difficulties that Apple had working with Sun. Talk about that whole thing. Why did Java not really pan out, maybe, for Apple?

Naroff: Well, I think at the highest level, Java is about a platform. It's-- I know as a tools guy, I like-- and a language enthusiast, I like to maybe gravitate towards the language and say, "Wow! It's cooler than C, because it doesn't have header files, doesn't have to deal with the pre-processor, is much cleaner in many regards. It has garbage collection from the beginning. So if you just look at it from a language perspective, there's nothing to get too upset over. But Sun was not really all that receptive to separating the Java language from the Java platform. So from their perspective, if you wanted to have a Java language compiler that was a native compiler, but didn't support any of their APIs, they weren't interested in that. They wanted it to be indivisible. They wanted the platform to succeed. They were adamant that the platform was where they were going. And that's an OS, right? That's an operating system. And Apple

is in the operating system business. Right? So while we were smitten with just the language, we had a big clash over the degree to which the language could be separated from the platform, and whether we could ever, you know, use it with our platform without their platform. And they just were never interested in that. Like for instance, when they were thinking of extending their platform to deal with video, we had made an overture that, you know, "We have QuickTime, it's wonderful! Why don't you adopt our QuickTime APIs? Which are proven and we're always driving them." And so we were trying to find a collaborative place to work with them where if they're so adamant about their platform, and they're not going to give it up, at least work with us, right? And that was never happening. They, at the time, they really had a very big ego about where they thought their platform was going. They were convinced-- they were really, really convinced that their platform was going to take over. And we-- so we weren't in a great bargaining position with them, and Apple was still in a place that was unclear. Things were looking better, no doubt. The company was, you know, from '97 to 2001, was definitely doing better. But it, as you know, didn't explode until iPods and iPhones and all the mobile stuff. Which ironically, is the domain that Java originally wanted to target, right? More embedded devices like that. So I think it's yet another example where unless you have hardware for a platform to show its energy, and to show its prowess, and to show how great it is, it's really hard!

Hsu: Or a killer app at least.

Naroff: At least. <laughter> At least. So I think they were drinking a little bit too much Kool-Aid, and eventually Steve just got bored with us trying to work with them. And that's when we started divesting and just worrying about ourselves and just not caring about Java all that much. Obviously, we-- Apple still supports Java. There's still a Java on the system. I have no idea how much Apple does, and how much Sun does. Clearly, Apple is in a much more strategic place, where if they said to Sun, "We're not supporting Java," Sun would. Well, actually Sun? Oracle, right, because Sun doesn't exist anymore. Which, again, is ironic.

Hsu: Actually, I think Apple no longer does its own Java. I think you have to download it from Oracle now.

Naroff: From Oracle! There we go! So now the world makes sense. Right? The world makes sense because, in fact, that is exactly what should have happened all along! My team and all the work we were doing was a concession that Apple needed to get strong again. And until Apple was strong again, we had to hedge. And you know, I wasn't savvy enough, or no one predicted the type of success Apple ended up having. Right? If you knew that was going to happen, we would have all behaved differently. But you know, so it's very, very interesting how Sun stumbled big time. But I think one of the lessons someone can learn from that whole episode was that it never helps you to be so arrogant; and Sun was really arrogant. Because Apple still had an amazing brand. And it just was a shame that Sun didn't respect Apple enough to work closely, make concessions. Because I do think Java, the language, might have had more traction if they were more flexible. I don't know. I mean, I can't prove that, but I do think it was an obstacle, their whole attitude.

Hsu: Well, yeah, Java on the client side never went anywhere. But they did do well on the server. <laughs>

Naroff: Yeah, and that's fine.

Hsu: They're still pretty dominant in enterprise.

Naroff: And that's fine. And I'm sure they have a server Java, that Oracle develops server Java for Mac OS X, right? And who knows? But yeah, I don't-- the biggest thing, as I say, that was regretful is that all those resour-- that hedge was at the expense of the tools. Which is why Steve ended up agreeing that in '92 at least-- wait, what-- wait- wait--

Hsu: 2000-?

Naroff: Why do I keep on going back to '92? What is '92? That's yeah, that's NeXT.

Hsu: That's when NeXT--

Naroff: In 2002. Yeah, I gotcha. Too many dates. And we've been going a long time.

Hsu: Right. <laughs> So that's when Steve agreed that-- you were saying--?

Naroff: Well, Steve regrets that we didn't do more with Metrowerks, to help cover that time period where we were investing so much more in Java, at the expense of the tools. But it doesn't seem like, at the end of the day, it really mattered that much, because we had a good relatio-- unlike Sun, we had a really good relationship with CodeWarrior, and they ended up making the transitions that we needed, and the processor changes, and of course, there's so many things that are dependent on the tools. And anytime the OS or infrastructure changes, there's a likelihood the tools might need to rev. And so-- and I take some credit for that, because again, I worked really well with them, and coddled them through some of that, along with Developer Relations, Clent Richardson and his team. So it was a team effort. But we certainly didn't act arrogantly. We respected that they had our developers in their hip pocket.

Hsu: Right.

Naroff: Yep.

Hsu: And of course, that was to support the Carbon transition, right? Because the Cocoa developers were still using Project Builder from-- the NeXT version, or the old NeXT Project Builder.

Naroff: Correct.

Hsu: Right.

Naroff: Correct, but we wanted to get the Carbon people over to our newer-- the new APIs, so them supporting Objective-C to support that transition was much easier for their developers than asking their developers to go to more UNIX-centric tools, and so on.

Hsu: Right. So the goal was to have CodeWarrior—[so that] you could program Cocoa using CodeWarrior. That was the goal.

Naroff: Exactly, exactly.

Hsu: Right, okay.

Naroff: And they did modify their compiler to do Objective-C. Oh, yeah.

Hsu: Right. I just don't remember if anybody actually used CodeWarrior to program in Cocoa. <laughs>

Naroff: Well, chances are you weren't hanging with that crowd.

Hsu: That's probably true.

Naroff: I don't think we know.

Hsu: Right.

Naroff: And I can't say I had great vis-- or remember exactly how many people were doing it. But independent of that, it needed to exist for it to be even be possible. Because the other proposition of ditching CodeWarrior just to use a couple of our APIs, if that's what you needed, that's a big--

Hsu: So it had to be a big checkmark on the business end, so that people could justify--

Naroff: It's an enabler.

Hsu: -- sticking with the platform, essentially.

Naroff: Exactly.

Hsu: Right.

Naroff: Exactly. Because the platform was-- you know, the number of Cocoa boxes out there still wasn't vast. I mean, there were still a lot of developers that, you know, either would threaten they're not moving their code over, or ask Apple to finance their development, for an awful long time! Until we gained traction. Like now it's wonderful, obviously. Apple has so many developers, because they sell so many devices. But before all that success-- developers were an angry mob that just didn't want to do any work without proof there were millions of dollars at the end of the tunnel.

Hsu: Mm hm. All right! I guess we'll pick-- we'll end there and pick up when you start working on Xcode next time.

Naroff: Great! Thanks!

Hsu: Yeah. I guess I'll just-- maybe I'll just say at the end that, like you mentioned the iPhone. Sort of the irony behind all this is that the iPhone ended up driving up the adoption of Objective-C beyond <laughs>-- way beyond what it was before.

Naroff: Oh, yeah! Well, it was proof that Steve was right when he said, "All we have to do is sell more units, and believe me they'll flock to the language and the APIs." And clearly they did. And clearly the iPhone offered a compelling experience. So yeah, the apps out there are just incredible. Yep.

Hsu: Mm hm. <laughs> Thank you!

END OF THE INTERVIEW

