Oral History of Steve Naroff, part 2 of 2

Interviewed by:
Hansen Hsu

Recorded October 10, 2018
Mountain View, CA

CHM Reference number: X8800.2019

Hsu: All right. It is October 10th, 2018, and I am Hansen Hsu, curator, Center for Software History, and I am back with Steve Naroff. So last time where we left off, we were talking about Steve's work on Java at NeXT and at Apple, and so let's-- we're going to move on to the next part of the story from there. But, Steve, you mentioned you wanted to rewind it a little bit.

Naroff: Yeah. In 1997, when or after NeXT was purchased, I had come on board to-- I guess the title was director of Java Technologies and Core Tools, and I was in temporary housing while my home was being finished and got up to go to my first meeting at Infinite Loop, and it was a really exciting meeting, because Alan Kay was going to be meeting with me and Steve Jobs and Scott Forstall and Bud Tribble regarding some work that he was doing, not so much, I don't think, to sell us on it but to educate us on it, and because I had never met Alan Kay and as we've talked he was a big hero of mine being a luminary in the object-oriented community, this was a really big thing for me. So [I] show up at the meeting, and Alan was there. We all shake hands, and before the meeting got going, Bud Tribble looks across the table and says, "Steve, you look like you're having an allergic reaction to something." I'm like, "Really?" because I felt fine. He goes, "Yeah, there's marks on your face," and I said okay. He said, "Are you taking any medications?" and I said, "Well, yeah, I'm actually taking an antibiotic, because I had a chest infection," because we were moving cross-country from-- I actually started taking it in North Carolina. So he said, "You should go to a hospital," and I thought that was a little reactionary, but I said, "Well, I'm not going to go to a hospital, but I will go home and see if-- how I'm doing," and so I went home to the temporary housing, had to leave the meeting, really bummed that I had to leave the meeting, and took some Benadryl just to see if it could basically stop whatever supposed allergic reaction I was having, and soon after my health went very south, and they had to call the paramedics. My wife was with my two kids. I believe they were two and three or three and four. Paramedics come. Blood pressure was 60 over 20, serious stuff. Turns out that I had a severe allergic reaction to the antibiotic and ended up having something called Steven Johnson syndrome, which is a medical reaction where your skin, worst case, burns off your body. So I was really happy Bud had the wherewithal to tell me to leave the meeting and go to get help, because Bud--

Hsu: He went to medical school.

Naroff: …had a medical degree. He was a brilliant computer scientist as well, but he had a medical degree, and he knew-- if Bud wasn't there, who knows what could've happened if I wasn't treated, because 60 over 20 is pretty bad blood pressure. So, long story short is I was in the hospital I think only a couple days. Fortunately I didn't have the most-severe reaction, but my skin felt like it was on fire for, I'd say, almost a week. I had to bathe in this weird substance that they told me, and I have almost no recollection of it because it was so traumatic. So now I know I'm allergic to sulfa-based antibiotics. I will never take them again, and so that was an interesting introduction to my new role, almost dying. So it's good we can laugh about it now. So then I settled into this job, and as-- we talked about Java quite a bit, but one of the other responsibilities was to merge the compiler teams. There was the Mr.C compiler team, and there was the GCC compiler team, and, as we've already talked about as well, there was the CodeWarrior compiler, and one of the things I failed to mention before that I think's an interesting side-note is not only were Apple's third-party developers all smitten with CodeWarrior, but Apple's internal

developers were also dependent on it for building many of their projects, which was really verboten to the Steve Jobs/NeXT culture. So it puts a little more meat on the bones of why I was advocating a much closer relationship with CodeWarrior. The other option was obviously to wean them off and have them dependent on Mr.C at the time, which was the dominant PowerPC compiler. It had a very nice PowerPC backend. GCC's was good but not great. So we evolved both backends and so on, but the big challenge there was-- and this was another thing that was quite distressing to me as the person leading the effort was I found out how so many engineers don't like learning new code bases, don't like sort of leaving their compiler behind and gravitating towards new code bases. It was something that I never really got used to, because, as you could see with my career, I always like learning new things. Every one and a half to two years I was doing something different, and to me, basically working on the same exact code base for, let's say, 5 to 10 years is like playing the same tune for 5 or 10 years. Wouldn't you want to play a new tune and learn something new? My patience for some of it at the time was short, and so that's where some of the challenges were. But, yeah, I did that job and managed the Java effort for about four to five years, and it's interesting. I was reminded of this while preparing for our talk. We didn't actually ship the 1.0 version of Mac OS X until 2001, and-- which was roughly around the time I started really getting antsy with being a Director, at the time, actually, a Senior Director I had been promoted to, and wanting to get back into doing hands-on engineering. I was working for Avie at the time, and Avie was, among some other things, a little impatient with, let's say, my lack of wanting to do performance appraisals. I really liked mentoring people, and I liked writing performance appraisals, but I had a pretty large group at the time, for me, at least, and writing performance appraisals was not really something that-- I'd much prefer to write code, design code. So I did them, and I realized they're important, but I realized that e-mail, outbound communication, giving talks, which was also new to me-- I mean, I had given small talks at NeXT. We had a pretty-- a small clientele compared to Apple. So getting on stage with 1,000, 2,000 developers was something I did, and I liked the educational part of it, but it wasn't, let's say, what I was born to do. So I lobbied Avie to hire my replacement so that I could get in the trenches and make an impact on the tools product, which included compilers but wasn't limited to compilers, so that I wasn't spending all my time managing. So…

Hsu: So then you hired your replacement, Ted Goldstein, correct?

Naroff: Yes. Ted was someone who had--I think he had worked at PARC years ago on Smalltalk. He certainly had a name in the Smalltalk community, I think worked with Adele Goldberg there, and was at Sun at the time. So we hired Ted from Sun. Ted and I, I think, had met at an OOPSLA, knew each other. At that time period, it was hard to find my replacement. I mean, not only was Apple skeptical that the company was going to be turned around, but most people looking at the company from outside were skeptical as well. At NeXT, well, when we founded NeXT, we had our pick. Everyone wanted to work at NeXT, at Apple not so much, which, in retrospect, is kind of interesting, given Apple's success obviously far exceeded NeXT's success just as a company. So, yeah, it wasn't easy, so that's why we pretty much settled on Ted. When I say "settled on," I wanted to give Ted the shot, and that's why I basically asked him to come in to talk, and he spoke to Avie and, I think, did well with the interview with Avie. The interview with Steve did not go well. Steve told me he did not want to hire Ted. He saw something that we didn't see, and I told Steve that I wanted him to do a second round, and Steve did do a second round and wasn't a wild advocate after the second round but said, "If you and Avie think this is the right thing to do,

then you should do it," and we did, and it-- for me it worked out, because it gave me the freedom to work on Xcode and many other things, which we'll talk about. But it was interesting that Ted had a rough time adapting to the Apple culture. Actually, Ted had come in as a vice president. He had asked to be a vice president, which I think is one of the things that bothered Steve and why Steve's antenna went up. Steve liked tapping people on the shoulder to be managers, leaders. Steve was a little more skeptical when people were doing it themselves, and I was in management, I won't say begrudgingly, but sort of like a community service. It's like, okay, I have some really great engineers here. They really don't have interest in it, and my communication chops were pretty good, and so I just decided I'll do it, and it just-- the big difference is that at NeXT I had a four-, five-person team and could still have the hands on the keyboard. At Apple I was managing 60, 70, 80 people, and there was no way I was able to actually contribute while I managed. So, yeah, we hired Ted, and I did whatever I could to help Ted ease into the culture, and I think we were somewhat successful.

Hsu: So let's get to talking about Xcode and maybe what led up to that. So we had talked about ProjectBuilder, the creation of ProjectBuilder, last time at NeXT, but I remember also that there was a period where there was a-- internally there was a Project Builder X that was-- replaced the original NeXT version of ProjectBuilder, which became ProjectBuilderWO, for WebObjects?

Naroff: Right. That's right.

Hsu: I remember that, and that was sort of like the basis for what became Xcode, as far as from what I remember.

Naroff: It did. I believe you're correct. The big difference was or the big issue of the day was it still looked like a NeXT-style app, and given the other UI changes that were being implemented to basically rebirth a lot of NeXT technology in the context of Apple, which involved many, many people throughout Apple, Project Builder even in the form you're talking about looked like-- it didn't look like a first-class, let's just say, Apple product from a UI perspective. So I had been using some of the newest UI stuff outside the Tools team that was going to be released and had the very basic insight of "I want to make sure Project Builder is leveraging the same idioms and looks like a smooth continuum from the user system to the developer system," that it didn't look like-- it had-- it was as modern in every way-- which included really nice icons, that kind of stuff, that it had the fit and finish that Apple's famous for, and I had pitched that to Steve, and Steve was very supportive, and Steve gave me some of his top UI guys to collaborate with, and I was supported by Ted in picking, I think, four people from the Tools team who were like the A-team to work with the UI designers. So it was like I was a Director with my hands more on the keyboard, where the Director was more like a hands-on Director as opposed to a management person. I had lots of opinions on what the-- how the UI should behave, how searching should work, eventually what code sense would become, because DevKit, the tool that we talked about prior, I had ideas for how that would make the editor smarter so you could get code completion, stuff like that. We went off for, I'd say, about four months and prototyped that…

Hsu: This was what year, 2001, or…

Naroff: No. This was, I think, 2002.

Hsu: 2002.

Naroff: 2002-ish. Yeah. No, 2002. I don't think we shipped it until 2000-- I think maybe-- was it? I don't recall whether it was in-- I guess it wasn't Leopard. Leopard might be the release, and I don't even know..

Hsu: That sounds very late. Is that..?

Naroff: Well, Leopard was when? No, no, Leopard's way in-- you're right.

Hsu: Leopard's 2006.

Naroff: '06, yeah. You're right. Well, I…

Hsu: Which OS release did it ship in, do you remember?

Naroff: I should've got the…

Hsu: Panther or Jaguar?

Naroff: Yeah. I think the talk that was given at WWDC was 2003, so whatever 2003 coordinates with.[1]

Hsu: That might've been Panther. I think that might've been Panther. We'll go back and check.

Naroff: Yeah. So we had done a prototype, and it was far from done, since four months' work certainly wouldn't be enough to actually make the full product. There was a lot of work to do. So we arranged a meeting with Steve Jobs and Phil Schiller and Ted, and I forget exactly who else was there, but those were the main-- Avie was there. That's right. So we had lots of vice presidents in the room, and I gave Steve the demo, and Steve was thrilled, absolutely thrilled. What I thought was going to be a 15-minute demo ended up being a lot longer, because there was a lot of talk in the room. I think it lasted quite long, and then-- and this was mainly UI. There's lots of other things we added that we'll get to, but at that point it was mainly UI, because that's what Steve cares about, typically, and Steve had said, "Well, we have to demo this at my keynote at the developer conference," which was great news. I was like, wow, that's great, and he said, "Who should demo it?" and since I had just given a pretty good demo, I was feeling great. I said, "Well, I'll do it," and he said pretty quickly, and I'm not so sure these are the exact words, but, "Oh no, you're not good enough," and I was like-- because it's a room of VPs. I had just done this great thing, and he's happy. Well, that was terse, "I'm not good enough." So I shut my mouth, because it wasn't worth getting into it with him there, and he said, "Let's get Chris Espinosa," who was, as you know, with him in the garage, still at Apple today, and Chris worked for me I think at the time working on AppleScript. But Chris was a great presenter, and, yeah, he's very charismatic on stage and just a great

[1] [Editor's note] Panther in 2003 is correct.

speaker. So I said, yeah, cool, so let's ask Chris to do it. Okay. I was quickly past the hurt feeling. So the meeting ended. I went back to my office, sulked a little bit like I usually do when that type of thing happens, and I decided I'm going to pick up the phone and call Steve. So I called him, and his-- he had someone that was a dispatcher that would find him, so she picked up, and she said, "I'll find him, no problem." So she patched me through. He was driving, and I said, "Steve, why'd you have to do that? I mean, what was that about? You were happy. I just worked my butt off," and he said, "Listen, Steve. I trust you with engineering. You're a great engineer. You've been a great manager over the years. You're not a great presenter," and he said, "It's my stage, and I need to make sure I have the best person there. You've done great stuff. I'm not taking away from that," and I said, "Well, that's great to hear, Steve, and I agree with you. I just wish you would've said it like that in the meeting," and he just said, "Well, they all know you're great, and so I didn't have to say it." So that was that, and it-- looking back, there's no doubt that sometimes I'm sure I was a little thin-skinned, but when you're in the bomb run, the trenches, whatever you call it, and you're working really hard, it's tough to be talked to like that. But what was great about Steve and why we always maintained a great relationship and why to this day I have nothing but great things to say about him is he understood and, in fact, then obviously patted me on the back and made me feel good. So he was someone that respected that I picked up the phone rather than let it linger or hold a grudge. I think he was happy that we basically cleared the air. So we were given the go-ahead, so we went back to engineering, and like a lot of things we had, it was sort of secretive. Not everyone in the Project Builder team knew what was going on. So then when we rolled it out to the team at large, like a lot of things, there were mixed feelings. There were people like, "Well, why can't we just stick with this or do that?" There was angst, and I guess similar to what I was saying before about compilers, no matter what you do in an organization, change is always met or often met with grumbling. So there was grumbling, and I had to live through that and work through that, and it-- I don't really recall it was all that troublesome, but that's part of the stress of being in that environment and trying to push things, and one of the reasons I had done this job and could-- and I was sort of one of the few people that was able to do a job like that is because I had the respect of enough people to push for-- an agent for change, let's call it. No doubt that not everyone loved me or the situation, but it didn't really matter. We needed to move Project Builder forward. Then the UI was in the hands of Steve Llewellyn, Anders Bertelrud, who were working very hard on it, and I forget everyone's name, but those are two that I certainly remember. I then went off to figure out how do we make compiling faster so that even if we aren't as fast as CodeWarrior by every metric, how do we improve this, because people are going to-- I knew people were going to say, "Well, yeah, it's prettier, and, yes, some of this functionality's cool. We get it, but the compiler's still slow. When I make one change, it still takes too long," and that was really up my alley, because I'd written several compilers. I had worked on GCC for years. I knew what the limits of GCC were. So one note of good news: they had implemented precompiled headers in GCC. At NeXT, I had to implement our own version, because Richard [Stallman] was unwilling to focus on that at the time. But this was five years, six years later, and I don't recall who did it, but they had precompiled headers that worked with C++, which was a limitation of our other scheme. Our other scheme only sped things up for Objective C, which in the NeXT world wasn't all that bad, but more people in the Apple world were using C++. So that was sort of good news. Point 1 is that GCC had caught up to what many other compilers were doing to solve the header file problem. The other problem was link time, because if you're making one change to a module, I mean, if you have a hundred-module program and 100 files and you change every module, which is very uncommon, well, then it's going to take a long time. But the commonplace is you have maybe 100

modules, and you change 2 or 3. How do we make that really fast? And my idea for that was something called ZeroLink. ZeroLink was the idea of as soon as the compiler produces a relocatable object file, which is a .o file in UNIX world, that you can basically run the program from the .o file that has main, and from there it will lazily bring in every .o file that's needed to run the program, and that was enabled by position-independent code, where you could trap on a function reference. It was sort of the Java model, right, where Java-- it wasn't-- Java didn't have .o's, but Java had class files, and when you're developing a Java program, all those class files are brought in dynamically, lazily. So we were basically taking a page out of that book but in an operating system like ours was still challenging. It took some work. So that was a nice speed-up. The other thing that I prototyped was distributed builds. That would make, let's say, the builds-- Apple's global builds faster so that when they start building the whole system, if you had a farm of 100 computers that were helping, you would push out work, which, again, was a page out of my Computer Aided Design at least at an architectural level that you can use the network and the CPUs on the network to speed up CPU-intensive things. Now, there's no doubt if you don't have a precompiled header, you're fairly I/O-bound, because you're grab-- you're touching the file system and bringing in a ton of files. But if you had a precompiled header and the precompiled headers were bootstrapped on each machine that you were compiling on, things were fast, and that work was a derivative-- I had-- someone in the group was tasked to go do that, and they did it, and they claimed it didn't speed anything up, and because I know-- I was very skeptical. I'm like, "That can't be. This makes no sense, so you've done something wrong," and I'm forgetting the details of exactly how it unfolded or why their data wasn't what it should've been, but I took Trolltech's-- Trolltech was a company who was developing C++ libraries to-- pretty successful development tools company some friends of mine worked for, and they told me about their distributed build, which I think was freely available. I don't think we paid for it, and I had hacked it to do distributed builds in the context of Xcode, and it worked great. I mean, I don't think it sped things up order N, where N is the number, but pretty close, assuming you had precomps on each one. So that was interesting, and I think those two things-- it didn't make it competitive with CodeWarrior, but it showed that we were moving the ball, and I think at that time people wanted to see a sign of life, and they were seeing it, and considering the modest investment that we had made, I think we chose pretty smartly at that time, and Xcode was developed over the course of a decade or more, and lots and lots of stuff was done, bringing Interface Builder into it. There was always a tension of how customizable the UI should be, because in general Apple's gestalt on UI is you shouldn't have too many knobs, and developers like knobs. So there was always that tension that maybe the first version-- and I totally believe it. The first version was naïve in some ways. So in no way was Xcode 1.0 something that was the end-all, be-all, but it did-- it put a stake in the ground, and I'm proud of it, and I think the people who worked on it were proud of it, and then the team grew it out over years, and it worked great, and around that time CodeWarrior was sort of drifting away. I forgot what year they were bought, but I wouldn't-- by Motorola. I wouldn't be surprised if it was around that time. So I think Xcode happened because we were finally shipping a consumer version of Mac OS X. The Java hedge was sort of done with. For several years we had held it together on Mac OS 8 and 9, and that worked out.

Hsu: Could you talk about maybe the rebranding of Project Builder into Xcode and the name? Where did the name come from?

Naroff: I believe Ted might've come up with the name. I'm not positive of that, but-- because those're things that I know when I was looking back in time on whatever e-mails and documents I could find, I know that that was-- that's one of those things you wouldn't write down, but I do think-- I believe we had some naming contests internally, and I forget what name I liked. I don't think Xcode was my favorite, but I thought it was good. The Xcode icon had also many iterations, and Tim Lasko [ph] I think worked on that.

Hsu: <inaudible>

Naroff: Do you like the name?

Hsu: I like the name. But everybody agreed that it needed a new name is the thing, though? No?

Naroff: No, everyone didn't agree. Oh no.

Hsu: Okay. So how did that go?

Naroff: That was part of the tension with the team. Yeah. I mean, it's just like modern syntax. You had some people that were really religious about keeping the Smalltalk-like syntax, and then you had other people who were religious on the other side, and changing the name for some reason is an emotional issue for some people. I didn't care. I mean, I don't know what it is about my personality. I just didn't care whether we called it Project Builder or Xcode. So I let that sort of stuff just happen, and this is why I probably don't have a great recollection of how it settled. But I do remember there were upset people, and, see, that's the thing. As someone who was a manager, leader in various different roles, if you let all this stuff upset you, you just pull your hair out, and my hair was falling out anyway. <laughs> So, I didn't need that.

Hsu: <laughs> Okay so, after Xcode shipped, then did you work on Objective C 2.0 next, or what was the next thing? Or did you start working on Clang? Which was the next thing?

Naroff: No, I think after Xcode, I was-- as developers were bringing large apps over to Mac OS X, which was probably 2003, one that I remember is a 3D modeling app called Maya. I don't remember whether Pixar was using Maya, or whether there were just big clients who wanted Maya. But Maya was this really wild app that had a huge C++ code base. And they came to Apple sort of reading us the riot act because I think their app took like six minutes to load.

Hsu: Whoa.

Naroff: I don't think I'm exaggerating. Let's face it, even if it's three minutes--

Hsu: Yeah, that's a long time.

Naroff: That's still not good.

<laughter>

Hsu: Yeah.

Naroff: So, I was tasked as-- at the time, my title was Chief Technologist for the Tools team. So, while, on one end of the spectrum, Xcode was something I had worked on, on the other end of the spectrum is, we have some big third parties that need some love. Can you get in the trenches and figure out what's going on here? So, I did. And I don't remember the exact numbers. I just don't have them because that type of information typically-- when I left Apple, I took very little with me as you can imagine. I just don't-- I do have a letter from Alias|Wavefront that said how happy they were. But they didn't put numbers in there. Unfortunately, they just gave me a shirt that said Maya, a hockey jersey, because I think they were a Canadian company. But my recollection was we went from let's just say three minutes, even though I think it was actually worse. I think we got it down to like fifteen-twenty seconds. And my recollection was that our linker/loader combo, we had invested a lot of money making efficient for Objective-C because that's what we majored in. The algorithmic bugs that existed in that domain were being tickled by C++ big time, big time. So, C++, as you know, was implemented as a preprocessor. But even as the normal compiler, it generates symbol names that are just crazy long because it includes type information to implement overloading in them. So, a name that you see in the source code is wildly smaller and different than the names in the hash tables or the data structures that the loader has to deal with. So, I went in and figured out what algorithms didn't work well and prototyped the solution and then talked to Kevin Enderby, who owned that code. And he took it and productized it so that we had the benefit and maybe even fixed some other problems. I by no means fixed the problem, but I prototyped a fix and showed that yeah, we're messed up in this area. So, it was basically a linker/loader bug that was being expos-- bugs, it was probably more than one, exposed by C++. But it's a great example where-- C++ is a very complex language, and we were not using it all that much. So, the third parties were stubbing their toe, and so I walked in there. Pixar had similar things. I don't think they were exclusive to load time. But Pixar had C++ specific bugs too, which I worked at Pixar for some period of time to help them as well and brought back solutions. And the companies loved getting love from someone in the Tools group as opposed to just someone in the third-party support group, which clearly folks like that-- I mean it's possible someone in a group like that could figure this stuff out but unlikely. So, I remember doing some of that. I'm sure there was some other stuff as well. But then you asked about the 2.0.

Hsu: Yeah so, how did that effort get started.

Naroff: So, it's an interesting story. And my memory of some of this is not precise, but there's something you probably never heard about, which is always nice because I looked at-- I looked online. And even modern syntax is in the Objective-C wiki.

Hsu: Really?

Naroff: Yeah, it mentions that. It says that they never shipped it, but it's there, which is interesting. But what I'm about to tell you, I couldn't find anywhere. So, we, like Microsoft, had some Java envy. Not Java envy about where Java was going or apps that were being developed with Java, it wasn't that obvious.

We just had envy because they were able to get rid of header files. And they were able to eliminate lots of weird C constructs. They were actually able to punt some of the C legacy that we-- that bag that we always carried around. So, that's where there was envy. So, I don't know if you remember, but Microsoft had a language. I don't even know if they still support it. But it was called C#.

Hsu: They still have it.

Naroff: They still have it.

Hsu: I think it's important to .Net.

Naroff: Okay. C#, if you look at it, is Java. It's almost amazing that they got away with it. It literally is Java with a different name. Again, I really don't understand legally-- I'm sure Sun, [which] doesn't exist anymore. At the time, I'm sure Sun and Microsoft must have been-- their legal folks were probably doing a duel of some sort. So, because Microsoft had done C#, and we had Java envy, there was envy in Apple that why can't we do something like that? So, I started an effort. Well, I don't know if I started it or Ted wanted it started. I don't-- those are the things I don't remember. But it was called-- I know I was leading it, managing it, whatever, leading it, probably leading it more than managing it, called C*, not C#. So, C with a star, an asterisk. Right?

Hsu: It seems logical-- they have # <sharp>. We'll use * <star>.

Naroff: Again, this was an internal name.

Hsu: Right.

Naroff: If we would have had the bravery to go through with it, who knows what it would be called. So, one of the really talented folks I think I mentioned in one of the questions earlier was Mike Kahl, who was part of the Dylan team, I think we hired him. I don't think he stayed at Apple in another group. It's unclear whether he transferred, I don't remember, or we hired him. But Mike Kahl was in the group with Blaine Garst and other-- must have been maybe one or two folks who were working on this. I think Greg Parker might have been involved. And it was an aggressive proposal to get rid of header files, have immutable interfaces. By immutable, I mean not have interfaces that can be disrupted by the preprocessor, which is something I was just dying for. So, I was very supportive. But the reality at Apple at the time is if you couldn't do it on a release schedule, and you couldn't-- well, you had to do it on some sort of release schedule. And usually, the release schedule was a year, roughly. And even though there was a lot of interesting thinking and writing and collaboration on what this language could be, it started becoming clear that-- and I think around that time, Bertrand might have been managing the software group, roughly. I think he took over from Avie maybe in what, 2004-ish? Again, we can pin that down. But he, as leader of the software group, as you know from him being involved from the very early NeXT days, really enjoyed language design, and language-- he had lots of opinions on languages, whereas Avie was more of an OS guy. And Avie was fairly hands off as far as the language goes. In fact, Avie was fairly anti-garbage collection for most of his tenure. And as soon as Bertrand took over, Bertrand put garbage collection on

our radar screen again saying that, in today's age with these other languages, how can we be competitive without garbage collection. And at a top line view, he was right. But he also knew from being a fabulous developer and hands on guy that introducing garbage collection in our environment was going to be tough. I don't think he knew how tough. But he wanted to see us give it a go. And we did. So, now I recall that I was sort of leading this language thing because, again, given my history with Objective-C, it was sort of a natural. I think then Bertrand had a heart to heart with me saying, "We have to reel this in. We're not going to be able to do something this high-minded." So, rather than think of this as C*, this rebirth of Objective-C, which, as you know, many, many years [later] was Swift. We aren't prepared to bite this off now. I agreed with him. And let's whittle down and focus on four or five things that we're going to do to Objective-C and brand it Objective-C 2.0. I was totally bought in. It was really-- given our culture in that period of time because we still weren't firing on all cylinders from a sales perspective. Microsoft was a major company. They could do this and fail, and it wouldn't mean anything to them. And Sun didn't have any legacy with Java. In C# were all people who just couldn't stand Win32 programming. So, any relief there was good, whereas people didn't have that same objection to the programming environment with just good old Objective-C. So, there wasn't that, "Man, we really have to rev this, or else people are just going to hate us forever." So, Bertrand asked me to get more involved and manage the language, the 2.0 effort. And I did. I think Blaine [Garst] might have been managing it for a brief period. And he agreed that he needed his hands on the keyboard for the GC effort because he was actually going into the Kit code bases and making changes to help them move. We knew if we implemented garbage collection and didn't help people throughout Apple, there's no way we would have done anything because it was complicated. Just to give you an example, I don't remember all the details. But there's a couple-- at least two broad ways to deal with garbage collection. One is you rummage through a memory looking for objects that are no longer referenced. But you have no compiler support. It's all runtime. And that's the type of collector that Java typically had. The compiler didn't do much to aid the collector as far as I know. And again, this is-- I'm speaking from information at the time. It could be that that's changed even in Java. I don't know. But at the time, it was I believe a generational collector. And it was all runtime bound. The other type of collector is to have the compiler do automatic reference counting, which Apple ended up doing later on because the Clang tools did static analysis. And so, that was something Apple moved to later. But at the end of the day, Apple did, by some definition, succeed with implementing garbage collection. Bertrand was insistent that we convert Xcode. We did convert Xcode, and the performance was not great initially. But we worked on it and worked on it. It eventually was very good.

Hsu: So, you're saying that the version of—Objective-C 2.0 garbage collection was a generational--

Naroff: Yes.

Hsu: Runtime garbage collector.

Naroff: It was. And I don't remember how-- Patrick Beard was working on it with Blaine. Both of them were excellent. One quick side note, when we decided to not do C*, it was gut wrenching to the people who were working on it. I think we probably spent maybe six months working on designing it. And the implementation didn't get very far because we really were designing it. And Mike Kahl was I think unhappy enough that I talked to him about going into the team that was doing indexing because Mike had

done Lightspeed C. Mike had plenty of other chops to use in other areas. I think he was just disappointed. He wanted to work on something new and shiny. And when it came to evolving Objective-C, he was not-- didn't have the same background with it and just decided, "I'm just going to go off and do something else and let the guys who have the most experience like Greg Parker and Patrick and Blaine do the Objective-C 2.0 work." So, that's a footnote. So, yeah, garbage collection, Xcode was converted. One of the problems with garbage collection is C because if you have a C structure, and one of its fields points to an object, the collector, unless that C structure is allocated in the GC heap, it won't find it. And not finding it means a leak and performance problems. So, all the C structures had to be combed through and dealt with in some way I believe to make that type of collector work. And we were stubbing our toe. And it was a very long process. And other features worth noting, we were moving to 64-bit, Intel. And we had an opportunity to fix some remaining problems with release to release binary compatibility. Just to explain what that term means, release to release binary compatibility means when you ship some code, the code that you depend on and code that uses you will not be broken if you change. And when it comes to methods, because of the dynamic dispatch, we were in wonderful shape. You could add a method to a superclass and not break your dependents, never any problem with that where C++ had lots of problems with that. Their virtual table layout did not accommodate release to release binary compatibility, which is one of the reasons these large systems that were being developed with C++-- like stubbed their toes is lightly stating it. They were really bothered by it because they-- it was just a very difficult problem. They'd have to pre-allocate slots just in case they wanted to, you know, add something. Those were early solutions. But it was very ad hoc.

Hsu: So, this was known as the fragile base class problem?

Naroff: Precisely.

Hsu: Okay.

Naroff: So, the fragile base class problem, since you have two things, you have methods and instance variables. Objective-C was perfect for methods and wasn't perfect for instance variables. So, we still had that problem where if a subclass added an instance variable-- a superclass added an instance variable, the subclass could break. So, we fixed that for the 64-bit transition. And it's a very good example that when you do superficial things to the language, life is simple, like change a syntax, add a qualifier, like public/private with a compiler. There's just so many things you could do that don't break binary compatibility. But when you talk about garbage collection, when you talk about release to release binary compatibility, fragile base class problem, these are of a different ilk, especially when you have shared code and dynamically loaded code. So, we had our opportunity, and that was also part of that release is to fix that because it's a new ABI, Application Binary Interface, for the 64-bit. Those programs weren't going to be running backward, only forward, which was true for garbage collection. As soon as you modified your code for garbage collection, you're certainly not going to run it on a system that doesn't support it. So, while it was a modest feature set, a couple of the features-- I mean garbage collection isn't modest, that's big. But the release to release binary compat-- that's modest at some level but very impactful at another level. And it was nice to fix that. We also added Properties for better setter/getters. That proposal was just-- wow, it just took forever for that to converge. And I was out with Bill Bumgarner,

who actually managed the Objective-C-- So, let me back up. When we were pretty much at the tail end of the ObjC 2.0 journey in terms of design, I actually took a medical leave of absence. I was out for a few months. And I handed the baton to Bill Bumgarner, who took the designs and stuff that we had and managed the team to actually deliver that. And I was out with Bill last night talking about some stuff. And Bill said that the Property proposal-- the only thing that changed from when I had left on the sabbatical there was that they went to WWDC and there were some objection to some of the syntax and semantics, without getting into the gory details, and that they went back after WWDC and revved it to incorporate feedback they were getting from developers. So, it was fairly well-defined, but there were certainly tweaks as they were refining it for final release. And like so many things, like modern, or like anything you change, there wasn't uniform agreement that where they ended up was the exact place to be. But I think Bill thinks more people were happy with the changes than with the original proposal.

Hsu: Right, one of the interesting things about the new Properties feature is it does allow for a kind of a dot notation kind of like modern syntax was going to do. And that seemed to be kind of controversial with some people in the community.

Naroff: Exactly. Well, some people wanted it to go further, and some people didn't. So, it was a hedge where-- and as you probably know, you don't have to use dot. It's optional. And the reason I think that makes sense is Properties, at the end of the day, are very simple named attributes. But when you talk about arbitrary keyword arguments, you're talking about some very, very long names if you were to transcribe them. So, I do think the dot made sense for that and not for message expressions in general. And as we've already discussed, I just really think the bracket notation in the context of C++ pays dividends. If C++ wasn't so dominant, I definitely think it would be cool to just go with dot if that's the decision. I would have no problem with that. And Bertrand-- Bill had reminded me. In fact, Bill was pushing back. Bill didn't like a lot of dot notation. And Bertrand loved modern. And Bertrand wanted modern is his recollection, and mine as well. And they supposedly had some knock down drag outs about it. Bertrand felt so strongly that he thought people might not pick up Objective-C if they had to use the Smalltalk syntax. I'll go back to Steve [Jobs]'s comment, which is, "Listen, Objective-C adoption is all about shipping units. When we ship units, people will use whatever syntax we provide for them." I mean let's face it, syntax is superficial. It's not semantics. It's just totally superficial. And I've met people who originally were not liking the keyword syntax, and soon after, like it. So, it's one of these superficial things that you get used to. And if it were semantics, if it were something like release to release binary compatibility-compromising or saying, "Oh, we can't do that. We will break apps," or, "We will do something horrific with GC, or slow down applications," those to me are substantive. Those are things that really hurt people.

Hsu: So, you shipped garbage collection, and it was used in Xcode. But it didn't seem like it got much traction beyond that.

Naroff: That's my recollection. And to be honest, I-- my recollection of it was that we really tried hard but never really shipped it. But my recollection was wrong. And it was shipped. But I almost think I stored away it wasn't shipped because, in my opinion, I was surprised if it was going to be used. I think-- I would love to know. It would be wonderful if we could know how many people used it. I'm sure people tinkered

with it. I have no doubt about that. But I'm talking about apps-- because just if you believe that so many of the apps like Maya that was C++, Pixar that used C++, if you start collecting all these things, the C++, the Objective-C++ folks had enough problems. They don't want to deal with that because to be honest, just like I was talking about the C struct problem, to this day, I don't really know to what degree we ran Objective-C++ garbage collected apps because we weren't using it internally. So, it was admirable that the team pushed that boulder up as far as they could. I really give them a ton of credit. It really was rocket science to get that stuff, to use Steve's term. But I'm with you. I don't really think many people adopted it. And as you probably know, I think two years later, it was deprecated in favor of ARC.

Hsu: Maybe a little longer than that.

Naroff: Was it longer?

Hsu: But it was--

Naroff: I don't think it was more than three, though. But-- well, those things we can look up.

Hsu: Yeah.

Naroff: Now, one of the strong people against garbage collection use, as the iPhone started being developed, Scott Forstall was dead against having any garbage collected code on the phone. And I totally agree with his perspective there because obviously, to fit-- Mac OS X was streamlined to fit on the phone. And Java too for their embedded devices like phones actually made an exception for garbage collection, which is really odd since Java started with such a strong position on garbage collection. So, I think for Java, it's a little bit more difficult to see them chop the garbage collector out. In our environment, it was natural since we had lived so many years without garbage collection. So, I think because Automatic Reference Counting is more deterministic from a CPU perspective because it's just adding some overhead to assignments, which is easy to-- you're not going to have a collector go out to lunch and not come back in a certain period of time. So, I-- Scott was very vocal about that. And Scott was very influential obviously within software.

Hsu: Oh, because he was also running Platform Experience on the OS X side as well, right?

Naroff: Yeah.

Hsu: Right, so he was doing both iPhone and OS X simultaneously for a period.

Naroff: Right. I really do think the critical mass of [App]Kit folk were "okay, well if Blaine and the tools guys want to go into our code and help with this stuff, but if they don't, we don't have the time and wherewithal to worry about this." So, it was sad in a way because it was kind of a point in time when we were getting big enough, and there were more and more projects and demands where this culture of always eating our dogfood was not easy to continue to drive, which is part of what made us so great. It's like everything we do in the tools area and the language area is going to be used big time. So, when

developers use forwarding or they use protocols or distributed objects, they're going to get all that testing that we are doing internally. And I guess it's amazing to me it worked as well as it did. But it-- of course, Xcode's a non-trivial app. And I believe Xcode shipped with garbage collection. But it's only one app.

Hsu: Yeah. Should we talk about Clang now?

Naroff: Sure.

Hsu: So, how did that get started?

Naroff: So--

Hsu: Or at what point did LLVM come into the story?

Naroff: I hired Chris Lattner in I believe 2005. And a fellow named Robert Nielsen who worked in the group actually pointed or put Chris Lattner on my radar screen. And when I looked at Chris's background and what he was doing, I was blown away because, to be honest, I hadn't hired many PhD students because most of them didn't have the hands on the keyboard as much as hands on writing papers. Not to take anything away from people who write papers, but Apple was much more oriented for people that had big ideas with writing code. And Chris's LLVM was open source as well. And so, I was able to look at it. It was much better than looking at the resume. And Chris had went even further to, I'd say, make himself attractive to me and to a company like Apple where he integrated his LLVM work as a backend to GCC. So, it was like just amazing work on so many levels. And we brought Chris in. And Chris, it was just apparent from the first fifteen minutes that this guy is an amazing developer, designer, person. It didn't take more than fifteen minutes for me to realize we've got to hire this guy. And so, that's how Chris wound up at Apple. And he was very soon after that put in charge of the compiler backends. One of his, I believe, first projects was to use LLVM to optimize OpenGL graphics code dynamically at runtime, very cool because compilers, unfortunately, historically have had a very static view of life. The idea-- they were command line tools. They take files in. They produce files. Those files are usually geared toward just running the program. The idea of iterative optimization that-- I mean it's cool to optimize stuff you can statically. But it's even cooler if you can then optimize the code dynamically at runtime, which for many graphics problems, makes a ton of sense. So, he did some work there that was really interesting. And the other non-trivial goal I put on his task is to actually compile the entire system with GCC as the frontend using his LLVM backend, which I don't know how long it took him, but it far exceeded what I thought it would take a mere mortal to do. Chris was just great. So, after he did that, since I'm a frontend guy-- I had a lot of experience with frontends. I won't say that I'm a frontend guy, but I had done a lot of work with frontends. And the DevKit code in the NeXT days was pretty much deprecated because it never did C++. So, I said, "You know, Chris, it would be great if we can finally make a compiler, a full frontend, middle, backend, that truly is library-based that is truly going to meet out compile time goals. And while GCC has been great for well over a decade, it's time for a new compiler that will support the IDE because, as we talked about just a little while ago, yes, we figured out some tricks to live with GCC's compile time, which was less than great. We wanted to roll our own, so we could basically solve problems like CodeWarrior was solving and lots of other things. So, I pitched starting Clang. And I told Chris I was going to borrow

some design patterns that I had used in DevKit. But fundamentally, that code base isn't going to be copied and used. I would take it to the next level in the context of LLVM. And so, Chris was really excited, and I then transitioned-- this was actually directly after my brief sabbatical, three-month sabbatical--

Hsu: The medical leave?

Naroff: Yeah, medical leave-- yeah, it was more medical leave than sabbatical, correct.

Hsu: And this was also 2005 or later?

Naroff: It was right-- I think it was in early 2005 or middle. It was only three months. So, you know. So yeah, I came back with this proposal to basically become an individual contributor, engineer, working for Chris. Chris had, I believe, taken over the compilers entirely. And Chris had implemented a pretty cool library-based preprocessor that was stream-based. And I was plugging my parser in to his preprocessor. And that worked out great-- that we called that Clang. The first LLVM developer conferences, you can Google it, I gave a talk that basically talked about the motivation for this Clang compiler rallying the troops and, in a lot of specificity, talking about things that we wanted to improve on when compared with GCC. A simple example is a lot of compilers will, when they give you an error, they will basically tell you what line it's on. And if you're lucky, they might give you a position within the line. But we wanted to be able to give ranges of exactly where errors were so that the IDE could do really smart things with potentially automatically fixing the errors because if you-- if it's a simple error and you have precise location information, you can rewrite it on the fly. So, code rewriting was one-- code rewriting, static analysis, performance, all of these things were dreams of what ended up being probably my fourth Objective-C frontend. I think four is probably right. But if someone said, "Oh no, it's really five," I-- maybe. But I really felt like I had finally reached the point where architecturally, there were no mistakes. It really was as good as I thought it could be. An example of something that was fixed, again I said it quickly, but it's actually pretty important, is being stream-based. A lot of compilers assume that they're grabbing characters from a file. And they don't know how to operate if you have those characters in a buffer. It sounds really simple, but it's an architectural aspect of most compilers that they just sort of assume they're working with files. And when you don't assume that but just have streams, you can do some crazy interesting stuff in memory with programs. For example, if you're in the editor and you want to do incremental compilation, you should be able to compile just a part of that program. And all those things were things we wanted to enable. So, I don't know how-- I don't recall how long it took me to do Clang. I would guess six months. And it was open source still. It was in the LLVM area. And we started to use it for code sense. We started to use it for language sensitive searching and lots of interesting things. And we hired a fellow named Doug Gregor to work on extending it for C++. Doug is another superhuman programmer that we found who was, I believe, working in the GCC community. And I believe I reached out to him. And he was interested. He turned that frontend into an Objective C++ frontend faster than I ever thought could be done, with quality. He was also on the C++ standards loop or committee. He was a serious C++ person and a serious programmer and I think is still at Apple today in the compiler group and just a really nice guy. It was a great team. And honestly, one of the reasons years past I had entertained and talked about buying CodeWarrior was the known difficulty of implementing a C++ compiler because there really are a small handful of people on the planet that can do what Doug Gregor did or Andreas Hommel at

Metrowerks. There's no doubt companies had probably try and hire teams of people and approach it with many people. But that doesn't fly with that type of problem. You need one person that's able to do most of the work, even if they're farming out little pieces to other people. And that worked out really well. And that was my last technical contribution before I decided to retire.

Hsu: Where did the name Clang come from?

Naroff: I know-- I believe Chris came up with it.

Hsu: Oh, okay.

Naroff: C lang.

Hsu: C lang. So, that also made possible the static analyzer in Xcode.

Naroff: Which Ted Kremenek did. So, we were-- we had a strong culture of oh, if we say it's going to do static analysis, let's prove it. And Ted was-- I don't know whether we-- I don't recall if we hired Ted because I think my recollection is that Chris brought Ted on board and that Ted actually had a lot of academic experience with static analysis. But when Ted first came on board, he wasn't doing it. I think he was getting his feet wet with some other projects. But yeah, once we did Clang, Ted was chomping at the bit to work on static analysis in the context of Clang, and did, and did great work. And I believe Robert Bowdidge, another engineer, worked on refactoring in the context of Clang, code refactoring. So, all of a sudden, that enabled some pretty sophisticated code sense, code refactoring, static analysis, pretty state of the art stuff. So, I really felt like my decision to step down as Senior Director, push Xcode boulder up the hill, solve some hard problems for third parties, push Objective-C 2.0 as far as I could before I went away for a little while, and then come back and basically start the missing pieces so that Clang and LLVM is the future, because I do believe they deprecated GCC. I don't know what year they did it, but I believe Clang and LLVM truly were the compiler. And in fact, now that I'm thinking of it, when I was talking to Bill last night, he says there's still a significant amount of Objective-C work being done within Apple.

Hsu: On the language or using the language?

Naroff: No using.

Hsu: Okay.

Naroff: Using, using, in other words, there's no grandiose thinking that the internal projects are ever going to be all pure Swift, which makes total sense. And it's really cool. See, Chris is the type of engineer that can do a really cool language like Swift and understand that it has to interoperate. There are an awful lot of talented people in the community that would not know how to do that or even be motivated to do it. They're doing this new grand thing. Why should I interoperate with this thing that started in the '80s? <laughs>

Hsu: Did you work on the blocks feature of Objective-C?

Naroff: I contributed a little bit of opinions on what it should or shouldn't be. But Blaine was the heavy on that, oh yeah. Yeah, I'd say my contribution, because I was coming close to that medical leave, was more in bringing focus and clarity to what the team should be. And I tried to get out of the way and let them each have their fun with the specifications of what each person was working on. If I saw something like with release to release binary compatibility that I knew was wrong, I would talk to Greg. Or if I had an opinion on blocks, I'd talk to Blaine. But I tried to give them space. I tried to take a step back because I believed the language didn't need to be pegged to one person having all the answers. I just wasn't a believer in that. And frankly, I took pride in the group solving their own problems.

Hsu: Earlier, you mentioned the iPhone and the effect that had on garbage collection. Could you maybe talk more about when you first found out about that, about the iPhone project, and how that may have impacted the compiler team in other ways?

Naroff: Not much, it was very secretive, very secretive. And that-- Apple's always been secretive. But it's always been particularly problematic for us because we're such a low level. So, with a processor change or something, we have to adapt quickly, or even with Xcode where they would change the UI and we'd have to then change the Xcode UI. So, yeah, we suffered a little bit. But we did not know much. We just had to react quickly as much as possible. And an interesting I guess sort of side note to that, one of the things that Avie and I used to butt heads over was keeping the Intel branch alive of the compilers and all the tools and the whole stack because he was-- because we had all that working as NeXT, he felt that one day, we may move from PowerPC to Intel and that we needed to have the tools ready to go. If we let them atrophy or let them just waste away, it might take too long, and we couldn't react. That was painful for me. That's why I'm mentioning in the context of pain. I know they're only slightly related and-- because I didn't have that big of a compiler team. So, I'm like, "Wow, we're trying to make the PowerPC backend. We're trying to integrate this. We're trying to do this." It seemed like wasting resources to constantly have the Intel branch tested and run. And certainly, my team squawked. But in the end, that's one of the areas where I believe Avie was right. And I was a good trooper at the time. We did what he wanted, and he was right. And he had to live with a little squawking from me. And I had to live with a little squawking from my people. But it's one of those things where it just showed how we really were a team. Some companies, some cultures might say, "Listen, I don't respect you, so I'm not doing this." We didn't have that type of culture. We respected each other. It didn't mean we always had to agree.

Hsu: I forgot to ask this question earlier, but-- so, we were talking about how C* morphed into the more modest Objective-C 2.0.

Naroff: Yes.

Hsu: In that process, what features got dropped and what features stuck other than I guess garbage collection was one of the ones that stuck?

Naroff: The big feature that stuck was garbage collection. The big feature that got dropped was immutable interfaces and any notion of deprecating headers. I also believe the original vision removed, let's see, certain types of pointers. And it was just trying to make the language more safe. And that's clearly not in the spirit of C. C is an unsafe language, and it's in-- the programmer has to basically live within a certain set of rules that avoid that. And they did, typically. Let's face it. When you're using Objective-C, just by nature, there's a whole bunch of programming idioms you're not having to deal with in C. And if you're choosing to use low level C in certain parts, hopefully you're doing it judiciously because it's performance oriented or memory oriented. You want to save memory. I remember when there was talk about making rectangles objects. And that would make rectangles much bigger because every object has a four-byte ISA, which is a pointer off to a class descriptor. And it's an example of where Objective-C is not appropriate for very fine grain objects like rectangles. So, as you know, rectangles aren't objects. But there are definitely people who are purists who think everything needs to be an object. And the hybrid nature of Objective-C for serious programs is more of a win than a loss. Again, the 3D modeling, or what Pixar is doing when they're modeling hair, or whatever they're doing, believe me. There are points in time when they are really doing crazy stuff to get space and performance even on, I imagine, today's processors. I mean let's face it. Memory, it's amazing how much memory is on machines now. And it's amazing how much disk is there. But the applications grow every year too, so I think if you're writing pro, sophisticated applications, it's hard to be lazy. And you don't want a language that stands in your way of optimization. There are so many applications. I'm sure there are tons of the apps on the [i]Phone that aren't performance bound. So, for them, they could be pure whatever. They don't have to ever use this stuff. But on the desktop or enterprise apps, C still has value. So yeah, we-- the big thing we abandoned is trying to give any semblance that C isn't the base language. So, we retained basically everything that Objective-C supported originally.

Hsu: Okay, and would the syntax have remained similar? Would it still have looked like Objective-C in the original proposal?

Naroff: That-- the syntax-- we never got-- because, again, that's sort of considered superficial, we never got to the point of nailing that down because even Dylan-- I think Dylan originally had-- the project Mike Kahl was involved in. That was Lisp-like. And then they ended up revving the syntax to be more traditional. And so, syntax is easy to change. So, no, I don't think that was a big focus.

Hsu: Right, and there were no plans to-- it wasn't even talked about. It was not in the radar of whether it would have a different syntax.

Naroff: Well, I'm sure there are some people in the company that wanted it to be on the radar. But I don't remember it being on the radar. We had proven with modern-- because that went pretty far, as you know, we had proven that, listen, it's obvious syntax isn't a big deal. If we want to do it, we should just do it. But the question of whether it would upset more people than make happy is such a hard thing. And as you know, Apple does not have a culture of, "Let's go ask our developers, or let's go ask our users. We'll do what our users or developers want." There are some companies that would create a survey <laughs>, send it out. Apple wasn't cut from that cloth.

Hsu: So, talking about this earlier attempt to replace Objective-C, obviously later on, we have Chris Lattner creates Swift, which actually does do that. It actually does get rid of the C basis.

Naroff: Exactly.

Hsu: It was originally-- there was a slide that said Objective-C without the C.

Naroff: Yeah.

Hsu: And so, how much of that earlier proposal or effort do you think might have influenced the later stuff? Or was there any connection?

Naroff: I doubt it. I doubt it. I mean Chris-- at a philosophical level, there was so many people that were sick of the C preprocessor, sick of header files, and wanted to do immutable interfaces. So, at a mindset level, there was a lot of commonality. But at a specific proposal level, no, I doubt anything, no. But really, it's-- I don't think there are many people that would defend C's wild C preprocessor. I mean Bjarne Stroustrup has been complaining about the C preprocessor since 1980 when he started working on the language. And I think he was probably thinking about, could he, given his position, constrain it. I mean he's written a lot of papers. And I'm sure we could probably find one. I know we could find one that complains about the preprocessor. Whether we could find one with a specific proposal for what to do about it is another thing. But he feels the pain. Anyone writing compilers and wanting to develop tools feels the pain because it's such a power tool, and it's such a performance bottleneck. But the preprocessor we wrote for the LLVM Clang project that Chris wrote was very high performance. And in fact, one of the things I was reminded by when I was doing some Googling earlier is that Clang and LLVM would actually-- had a persistent tokenized format where if, rather than have a precompiled header with gory compiler-like data structures, you just had the tokens. I remember we were shooting for that. You could just take the tokens and put them out to memory. And then if you have one large file, let's say Appkit.h, that may expand out to two thousand files just to take a number, if you make one file that's been expanded with all the tokens, that optimization alone is actually pretty amazing. And then you just take the token stream from one to n and process it and pump it through the parser. And that's actually incredibly fast. So, one of my big things when we were working on some of this is precompiled headers, there's so many ways to skin it. GCC and most compilers, what they do is they actually write the gory low-level compiler data structures to disk. Those on-disk data structures are huge. So, the benefit of just streaming off the tokens is, even though you're still having to do processing to derive data structures, in some cases, it might actually be as fast as reading those data structures off disk because the data structures are, to throw out a number, ten times bigger than the tokens. So, I think-- that's one of the things-- this is sort of totally random, but one of the early influences that Bertrand had on me was the concept of lazy evaluation was much as possible. And he was always pushing for that. And it really influenced my thinking on a lot of things. And lazy evaluation with today's processor speeds is just goodness I think in almost any area. I mean you could take-- you could even imagine a large Swift program that doesn't depend on header files and has immutable interfaces that you ship the source code around. I mean if the runtime system is fast enough, the source code may be a totally reasonable file format.

<laughter>

Naroff: But as you know, with C, you can't ship source code around because it's so dependent on the file system.

Hsu: So, how-- I mean was any of this Swift work begun while you were still at Apple, or--?

Naroff: No.

Hsu: No, okay.

Naroff: No, I left in January 1st, 2010 I believe was my goodbye date. And Steve [Jobs] passed away roughly a year-- I think 2011.

Hsu: Right.

Naroff: And his illness, which, as I believe you know, lasted quite a while. He lived with the illness for like seven years. Seeing his demise was hard. And I felt like it was time for me to hand the baton to Chris Lattner. Well, there wasn't really much-- I wasn't a manager then. So, I guess I'm speaking totally figuratively, right? But it was time for me to go away <laughs> and totally let him run the show, which he did for a while.

Hsu: So, you finished your work on Clang and then retired.

Naroff: And then retired, yeah. And yeah.

Hsu: And could you talk a little bit more about Steve's death and that time and your reaction?

Naroff: Yeah, I mean I did not have that much interaction with Steve during his illness. I remember-- wow, what year was this? Let's see, 2011 is when he passed. Like 2003 roughly, soon after he had learned about it, I remember bumping into him in the lobby and him talking about how he had no symptoms and that they only found the cancer as a side effect of a routine physical and some other tests they were doing. And he was pretty upbeat that because they found it so early that he was going to be okay. And certainly, considering it was pancreatic cancer, it's unheard of to last as many years as he did. But other than that, running into him and having that brief discussion with him, which I believe was fairly soon after the Xcode introduction-- one thing I was really proud of was that when he demoed it, even though Chris [Espinosa] demoed it, getting back to that period of time, he had mentioned my name specifically thanking me for the work, which was kind of uncommon and is tough because when you're mentioning one person, other people who are involved feel badly. So, I think he moved away from trying to do that. But in general, Steve really liked giving credit to the people who work on stuff. Years ago, didn't everyone even put their name on apps like in the Apple-- early Apple days?

Hsu: Yeah.

Naroff: Yeah. That went away. That was gone. But so, it's always cool to be recognized. And that's one of the reasons he-- internally, he would commonly recognize my team and stuff. So, that was good. But yeah, and I went to the memorial after he passed. And it was actually a NeXT specific gathering memorial. And it was like seeing family. It was a great event. And he-- seeing someone that's as much of a luminary as Steve, even his voice change, it's tough. It's tough because I mean he is just such a force, right? And when you think of someone that's such a force, you think they're immune to a long horrible drawn out demise. But it was one of the reasons I decided to retire. I just figured I had a great run. I was really happy with Clang and LLVM. They were-- the team was stronger than it had ever been. Apple was really starting to pick up a head of steam. And what happened after I left, the continued ascent as a company, all the people using Objective-C far exceeded my expectations or anyone's expectations. And I'm just proud that I was part of it.

Hsu: Right. So, some things that came out after you left, I guess ARC came out after-- Automatic Reference Counting came out after.

Naroff: Yes.

Hsu: The integration of Interface Builder into Xcode with 4.0, was that after?

Naroff: Yeah.

Hsu: That was after also.

Naroff: Yeah. Yeah, lots and lots of stuff was done.

Hsu: Right. I guess maybe the big thing that I'm interested in is what was your reaction to Swift when it was announced?

Naroff: It's about time.

<laughter>

Naroff: I mean yeah, I think "it's about time" was the biggest, and wow, "I'm so glad Chris is able to get the freedom to do it." So he deserved it, because he paid his dues there. And it's interesting. When I retired, there was a first year where I felt sort of odd about retiring. You know, but I think the transition I had made to engineer, and the various transitions I had made sort of helped. Because I mean, it's important to, I guess, be explicit, that one of the amazing things about Apple and companies that are run progressively, I'll say, is the idea of letting someone like me step down from a Senior Directorship, and lead a technical effort again, and then get back into management, and then go ba-- like the various transitions were-- I think are uncommon. I really do. And I mean, I think there are many reasons they're uncommon, but I think companies usually stand in the way of those types of things. But Steve really believed that people needed what he called the A-Team. And I know there's like-- he has his A-Team, and I know there's lots written about the hero / shithead rollercoaster, and I'm sure there's a pretty big

overlap between the A-Team and the hero / shithead people, right? <laughs> And but he really wanted to give someone the freedom to make the biggest impact. You know, if they would have said, "No, we don't want you to hire someone else to run the group. We like what-- we don't want to rock the boat. This is a critical time." But I think there was a real recognition that my talents were better used how I was sort of trying to drive it. And before Apple, I would just leave a company. I'd say, "Well, I want to do something else. I'm bored of this, or interested in that." But he knew that we were all volunteers, he always said that, "You're all volunteers." And that I really thank him and thank Apple for giving me the freedom to make those transitions. So but when I retired, I felt like, "Well, what transition is this?" Like, "What am I really doing?" And I mean, going from a high visibility job, whether it's engineering, management or directorship to being retired, psychologically was tough for the first year. Maybe even one or two. But after that it's great. For me, it was great. Because I believe programming is a sport for young people. I believe what I did is something that if you get 20 years, you should just be thankful. And like I see orchestras with lots of gray-haired players that are falling asleep. And I'm like, "Wow, can't you guys move over and let young guns in? Like my son." <laughter> And I think that it was really nice to be able to move over and let Chris [Lattner] excel. So I was done. I was done. And I had plenty of hobbies. I thought about coming out of retirement. Actually, the day that Steve passed, I was on a job interview with a company called Savant in the home automation space. I did one interview, I realized, you know, I wasn't in the mood. They wanted me to run their software group, and it just wasn't the right setting. I didn't want to move to Cape Cod. That's the other thing is all the moves, you know, going to Utah, then going to Connecticut, then going to California, then going to North Carolina. They say that moving is one of the most stressful events you can undertake, and I had done it a lot. And it took a toll. It did. Stress is not really good, and I was smart enough to get out when I could. And I really enjoyed retirement. So.

Hsu: I think one of the interesting things in the context of Swift, but also bringing it back to our earlier talk, our earlier conversation, you know, a lot of the work that you'd done on Objective-C and earlier systems was all dynamic. And Swift is kind of a step in the other direction. More towards static typing.

Naroff: So more like Java.

Hsu: Yeah. And some have called out there's a mismatch between the Cocoa design patterns and the way that Swift likes to-- you know, idiomatic Swift wants to do things because of the strong typing. What is your take on all of that?

Naroff: Well, I haven't looked at it to the degree these other people have. Does Swift have a traditional-- well, I should ask how much do you know about Swift?

Hsu: I played with it a little bit when it came out. I haven't done anything in 3.0 or later. So it's been a couple years, but I played with it a little bit when it first came out.

Naroff: Let's face it. The language influences the design of the kits, libraries, component libraries, ObjectWare, whatever you want to call, and they're influenced by the language. So it's not surprising that people see that mismatch to some degree. I think even when we wrapped Objective-C kits with Java, as similar as it was, there was still that mismatch. So it's probably, I would guess, similar, maybe not

identical, but similar to that. What would be interesting to me to look at is are there any pure Swift libraries out there being actively developed by either Apple-- because as we discussed, WebObjects eventually moved to pure Java, and Bill Bumgarner told me that that was a great product. He said it was better than the [Java/Objective-C] wrapper version. And so what I'd be looking out for by analogy is some pure Swift code, larger component libraries and compare and contrast where they believe they've simplified life for people. Ultimately if it doesn't simplify things, then obviously, it's not as compelling. Yes, it's obviously simplifying because they've pushed eject on a lot of C stuff. But other than that, where does it simplify?

Hsu: Right.

Naroff: Right. And I haven't looked at it to have a strong opinion. I mean, to prepare for this, I had my hands full with just remembering the last 30 years. I wish-- it would have been nice to do more work, but we spent so much time that maybe I'll look at it closer and have stronger opinions one day. Or maybe go out with Chris. I should give him a call.

Hsu: That'll be fun. Yeah. There actually are a number of web frameworks purely written in Swift. One of which comes from IBM, actually.

Naroff: Wow!

Hsu: Yeah. IBM's been heavily investing in Swift on Linux.

Naroff: Neat.

Hsu: So that's an interesting thing. But I don't think that they use anything like the WebObjects design patterns. And in fact, I think, you know, WebObjects and EOF and CoreData rely heavy on, I guess, was it key value observing-- key value coding, key value observing. I think those patterns are not well supported in Swift.

Naroff: Okay.

Hsu: Because of the strong typing. So.

Naroff: Well, another, like we talked about blocks. Blocks is another feature that-- it was added as you know for 2.0, and even though they may have introduced blocks with some newer APIs, I don't think the kits went through and added a lot of block support. I don't know.

Hsu: I think that the newer APIs--

Naroff: The newer ones do?

Hsu: -- did add a lot of block support. And then Swift supports closures as well, so. I think and Swift supports a lot more functional language type features like map reduce and things like that. So it's gone

more in that direction. And I think the other big thing that they've added is adding essentially categories on protocols. Extensions on protocols. And they've tried to push something called protocol-oriented programming in which you try not to use any sub-classing, and you try to do everything with protocols and extensions.

Naroff: Oh, that's cool!

Hsu: Yeah, so that's been a big push as well.

Naroff: Yeah, I mean, sub-classing was recognized as being less than great soon after objects were introduced. And there's no doubt that deep inheritance hierarchies are clearly like the wrong path. It's also clearly the wrong path to inherit from things that you anticipate changing. I mean, making a commitment to a superclass is a big deal, right? Because all your clients end up being dependent on it. Yeah.

Hsu: What do you think of-- looking at the way Apple's grown since Steve's passing, what do you think Apple's direction has been, and where it's going?

Naroff: Well, I believe Tim Cook has been doing a great job overall. When Steve passed, there were people who said, "Are you selling your stock?" figuratively. And I said, "No, I think Tim is the right person for the job and that I had total faith in Apple to innovate moving forward. I mean, I'm sure Steve is missed, obviously. But Steve knew, and often would say, you know, it's not about him. It's about the team. So I actually think that the area that is probably suffering more than anything is the culture. And eventually that will have a negative impact on the products. And I think some people that I've grown up with at NeXT and Apple would probably be less kind than I'm being right now. I think they already see where the culture is not helping them. I mean, obviously, they're not really putting, I don't believe, the A-Team on the desktop product anymore. So when I use the desktop product, I can find bugs and see stuff that, and lack of polish that you don't find on the iPhone. So you know, that's not too surprising. They're putting their energy behind mobile, which makes total sense. In fact, I have an older MacBook Pro that's really getting slow, because the new OSes are just, you know, need a lot more resources. And my son, who's totally cut over to a large iPad Pro with the keyboard, he influenced me to rather than buy another MacBook Pro to just go that route, which I did recently. And I'm trying. It's a little bit odd. I'm not totally there yet. But I could definitely see where that could be a very viable product and that one day Apple won't have portable computers and will just totally go that direction. I mean, in the music field it's pretty amazing what people are doing with iPads now with music notation and everything. And so many musicians just bring an iPad now in all the music. I don't know if it scrolls automatically based on what it's hearing, but you can imagine it doing that, rather than having a page turner. So it's-- what was-- what?

Hsu: I just, you know, what-- the direction?

Naroff: Oh, right, right, right.

Hsu: I mean, what do you think their biggest challenge is? What do you think they're not-- things that they could be doing that they're not doing?

Naroff: Well, yeah, I, in particular, would love to see more work on the home automation front. I have homes-- I've had homes and still have homes with Crestron, which is a big home automation company, and Control 4. And those companies, you know, have been getting better over the years. But it's still a very proprietary solution, very high cost solution. And so I've been looking for lower-cost solutions and Lutron has some interesting solutions for lighting, and Sonos is certainly doing some interesting things. And then there's, oh, what's that company? There's interesting companies for programmable remotes. And it's interesting because in the days of the big home automation guys, they would want to own the app where all these things integrate and they would even sell proprietary hardware to do that. But with iPads, the integration-- and phones, the integration point is the iPhone or the iPad. So if I have to run different apps to access those different parts of my home, I don't care, right? Doesn't have to be one vendor. So, and clearly Nest was a spinoff of Apple with Tony Fadell. I just think thermostats and lighting and remotes are high margin areas that I'm surprised Apple isn't get into more aggressively. You know? And in fact, the year after I retired, I was having a great discussion with a Genius at an Apple store, and when he learned my background, he was like, "Oh, wow! This is exciting! I'm talking to the guy who did this and that and the other thing." I'm like, you know, "I'd like to give back, and I wouldn't mind working here part-time," and I actually talked up-- I mean, this is now quite a while ago, this is maybe seven/eight years ago where I was talking up home automation and thinking Apple should get more into it, and he was agreeing with me. So I then tried to go through the retail hiring morass. Oh, my god! It's like dealing with the government! You know? They-- it was awful. So my sort of mini dream of, "Let me give back to the community, and maybe have some impact on the home automation strategy," the people they hire for the retail stores are mo-- they're-- it's just a different culture. It's a totally different culture, and even though there was some local support for, "Wow, we'd love you to come work with us," in general, the machine isn't oriented towards hiring old-timer R&D guys like me. But my interest in this area was born out of frustration with existing systems. And this was before Nest or Ecobee came out. So I mean, listen, I'm sure there are a lot of people having ideas about better home automation. But one of the things when I talked to Savant in the interview was, you know, "What are you guys going to do when Apple wants to own this?" <laughs> And they didn't have a good answer. But Apple hasn't been doing-- I mean, I guess Apple has their HomePod, but and I know that's software that probably does actually talk to some of these devices I'm talking to. And I'm not trivializing that. That's probably some good stuff. I don't have a lot of experience with it. But--

Hsu: There's a HomeKit framework.

Naroff: HomeKit framework. But you would think that the thermostats, which again, are high margin items Apple would want to sell. Because if they sell them, obviously, this HomeKit software integration is going to be better than working with Lucent, who basically has this Casita framework, where they're into things like just too much infrastructure. Right? I mean, this should be a simple hub. It shouldn't be a hub that has to talk to another vendor's hub. Because then everyone's on the network. So there just seems like too much complexity that Apple could solve if they got more serious about it. But, you know, to be honest, I'm not one of these people that get too animated about stuff like this. I just think, in general, Apple's doing good work, and I am happy that despite pushback, when he first did it, remember there was a big pushback when Tim announced the different sized phones, right? There were a lot of people that were saying, "Oh, Steve wouldn't do that. There will be one size. You have to buy this size phone."

Having different sizes is goodness, too, right? Different form factors. It just makes sense. Especially if you're as great as Apple is at manufacturing. If you can manufacture different sized phones and meet the demand, why wouldn't you do that? Right? The software platform's basically the same.

Hsu: What do you think is the biggest challenge facing the technology industry?

Naroff: You know, I really don't have any big thoughts. I knew that kind of question was going to come up. I will say that it's depressing to me the level of addiction there is to today's technology. So you know, it's one of the reasons I think Apple stock is going to continue going in a good direction, because people are so addicted to phones. They're so addicted to constantly getting a new model every couple years, on average. And to me, when I go to restaurants and see families all on their phone, not talking, I know Steve would also sympathize that it's too bad humans are so addictive to technology. And I think like when I go out with-- like I was out with my son's friend's father, who's a Vice President at Disney. And lives near us in Pasadena. We went out and I was so impressed that the whole night the phone did not come out! I mean, this is someone that probably does need to be tethered to his phone, because he has lots of responsibility. And the phone didn't come out at all. On the other hand, there are people on the other end of that spectrum that are always glancing at it, and playing with it. And I don't really want to be around people like that. So it's kind of sad that we helped enable something so profound, and it is profound! I mean, the level of functionality, utility that we get from these phones is just shocking! I mean, one of the apps, a very simple app that I love that I use in Pasadena or a lot of cities, where you don't have to use the little parking meter, where you used to have dimes and nickels and quarters and search around and have a thing of change. There's an app. You put in the zone. You put in your card. And you have a record of it. And when you're out, it'll tell you how many minutes are left. Now that's a simple app that I just think is groundbreaking. I mean, it's so simple, right? So you know, there's more and more stuff like that. So while on one hand, yes, I'm a little depressed at the addiction, I'm obviously animated and excited about what's happening on the phone. Hopefully, we'll figure out how to moderate. I'm also not a massive fan of social media. You know? I think it's polarizing and hurting the society. And like I posted on Facebook a little picture and copied you about this interview, and it was really cool because a lot of my friends now, when this is edited and posted are going to want to see it. So that's great. Or like if I post a picture of our kids. That's great. "Oh, look at your son!" It's great for that! But there's so much politics that goes on there, and I've been as guilty as anyone about posting political stuff and duking it out with people. And in the end, arguing with people you don't know is really a bizarre idea! Right? I mean, I've had people argue with me about some technical things they know nothing about, and they don't know who I am, and you just sit there like, "I'm not going to tell you who I am, or what I know, because I'm not that type of person. I'm just going to stop talking to you." And the same thing with politics. People try and argue with no knowledge about something. So yeah, I don't get why so many people love fighting over social media. And because of that, it's become very tribal, and again, will have a-- is having a pretty, I think, bad impact on the society. So simply put, I think less is more when it comes to using some of these tools. I think if that doesn't happen, it doesn't end well. So.

Hsu: What advice would you give to a young person starting out in the technology field today?

Naroff: So the field's a lot different than when I started. But I think one thing is hopefully still true is that if you're in a situation and you feel-- or you felt that you've learned all you can learn at a particular company, and you've stopped learning, and you feel like you're not working with the best people, make a change! I know that many people think that you make a change to get a salary increase. And I'm not saying that's bad. If people want a higher salary, and that's the metric they feel comfortable with, I can't sit here and tell them they're wrong. But I think it's better to look a little bit longer term, and look at what you're working on, and decide whether-- if what you're working on has legs and has a future. And not just what you're being paid, right? Because I think that in my career, all the transitions I've made, many of which we've talked about, it wasn't about money. It was about, "Wow, I want to learn more about objects. Wow, I don't want to develop software for hardware anymore. I want to develop software for software. Wow, I want to be around people who consider it more of an art. I want to work around people who motivate me and people who are better than me. And I want to hire people better than me. And I want to work in a place that doesn't shoot the messenger." And so there's so many cultural value, and yeah, money will eventually come if you make the right decisions. So I think while some people-- and one of the reasons I've never really been bothered with the fact that I really haven't gotten that much credit for Objective-C-- because as you know, Brad and Tom birthed it, and I raised it, I think at NeXT and later Apple, the people who care know. And just having my name out there, or writing a book, that was never really my goal to put my stamp on it, other than technical stamp, and make sure I was moving the ball forward. And Steve knew that. And I think it worked well for our dynamic. And also I think the other reason I lasted so long at Apple and NeXT was Steve really valued honesty. Raw honesty. He never wanted to be lied to, never wanted to be bullshitted. And a lot of people have a hard time not lying. And I'm not saying big lies. Right? I'm saying even a little lie like saying, "Oh, yeah, we're on track to deliver this like in the timeline you want." You know, if Steve came up to me, which he did often, as you can imagine, "Oh, how's this going? Are we in a good position to ship this?" I would often have to give him the not-so-great news. "Well, we're in good shape. We're 80 percent there. We still have this to do." He'd rather be told the truth, than lied to. And I think that was one of the big reasons he and I did so well together.

Hsu: Did he change over time, over the period that you worked with him from NeXT through Apple, that period?

Naroff: I think people like to refer to the NeXT Steve forward, as I think Steve 2.0. So I mainly knew Steve 2.0, and he didn't change much. I mean, in the early days of NeXT, he had a couple casual girlfriends that he would bring to the events. And it did seem like when he met Laurene, he became much more of a family man, and much more interested-- less interested in partying so to speak, and more interested in settling down. And he matured over the course of the years I knew him. But Dan'l Lewin and I were just talking upstairs, he was a hard guy to warm up to as a friend. You know, in the early days of NeXT I tried to ask him out to just go to music. And there was never any motivation on his part for doing that, which is fine. I mean, he did on one instance when I ran into him at a restaurant when he was with Laurene and Reed, when Reed was like two or three, was in a high chair, I was in Gaylord's, which was a fabulous Indian restaurant that he loved in the Stanford Park Mall, and I was just sitting by myself having dinner, I believe, and he noticed-- I actually didn't see him, but he noticed me, and tapped me on the shoulder, and says, "Hey, come have dinner with us!" So I did, and that was about-- and maybe a couple of times had been over his house, that's the most socializing we ever did. But I value that. At Apple, we

didn't do much socializing at all. I mean, but to be honest, all I cared about was that he cared about what we were working on and was in sync with it. Being his friend was never really a goal. He is someone that is hard to warm up to when it comes to friendships. Easy to warm up to with business, because he's so on top of everything. You know, I've told people that he was such a great communicator that he could talk about objects more elegantly than I could. Because he would listen to me and others in the NeXT software group, and run it around in his head, and I don't know, just tell stories about why it's so important that were way beyond the stories I could tell. I don't know-- have you seen "The Lost Interview"?

Hsu: Just part of it, yeah.

Naroff: So you should Google, they chopped it up on YouTube and have little snippets from it. But there's one part that's termed "Polished Rocks." And he tells a story about this old man that lived down the road who had this little tumbler, and he'd put in rough rocks, and run it overnight, and they'd come out all smooth and polished, and that was like Steve's metaphor for us working together, right? The software engineers, designers that we were all rubbing up against one another, essentially polishing each other's ideas. And he tells the story so heartfelt, and when I see it, it's just fabulous. So he's a great storyteller, technical storyteller. So.

Hsu: Was there anything we missed? Was there anything you would like to add that we didn't get to talk about?

Naroff: I think there was one minor thing. Remember I kept on going back to '92?

Hsu: Yeah! <laughs> Right.

Naroff: So a) my brain was mush at the end of the last six-hour marathon, and I did-- there was one event in '92 that I don't think had anything to do with what we were talking about, but I found an email from Steve, that I think is just--it's very simple, but interesting to note as the last thing, is that Steve was so interested in compensation and stock position being fair to people, that in '92 at NeXT, which is four years after I started, so I mean, I was fine. I was not having a problem with my compensation in any way, I don't remember being upset. But the salary and stock increases that were being proposed bopped up the management chain and Steve was just so hands-on, that he had to review everyone's stock and salary. And he sent an email to the senior staff, which included me, and copied HR and some other people, and he said, "You know, I agree with these salary and stock adjustments with three exceptions. Avie Tevanian, because of the additional responsibility he's taking on. Steve Naroff, because he's one of our best managers and underpaid relative to his peers," and then someone, Kathy Novak, who was in Pubs. And it's such a good example of him looking out for people, not only in terms of what they're producing, and being on top of them for their deadlines, but actually whipping out his wallet, so to speak, because he was running the company, and saying, "I want to give these people more to be fair to them." And it's funny, before preparing for this talk, I had totally forgotten about that. That's not the type of thing I would remember. But when I was looking through some of my emails and saved papers, it stood out. And just made me shake my head about what a mensch he was regarding even compensation, because there's so much negative that the media likes to write about him not giving someone stock, or not. And I

understand that he didn't treat everyone like that. And I'm just thankful I was one of the few that he called out. But I think an important side note is when you're Steve Jobs, or anyone who's running, at the time, you know, a couple really significant companies in Pixar and NeXT, having him discriminate is what you'd expect, right? So -- a lot of the people who complain about him not giving them this or him not giving them that, and the media hopping on that bandwagon, it's kind of silly. Because you well know in any company you're going to have people that receive perks and people that don't, and it's just great when a CEO is trying to actually pay for performance, as opposed to politics or other stuff. So it was really an environment that I found was free of bad politics. You know, the only politics were surrounding the politics of products, which meant, "How great are you making our product?" which is kind of a valid thing for him to focus on, so.

Hsu: Thank you!

Naroff: Great!

END OF THE INTERVIEW

