Oral History of Nitin Ganatra

Interviewed by:
David C. Brock
Hansen Hsu

Recorded April 24, 2017
Mountain View, CA

CHM Reference number: X8186.2017

Hsu: The date is April 24th, 2017. I am Hansen Hsu and I'm here with Nitin Ganatra.

Ganatra: Hello.

Hsu: Hello. So let's start with how did you get into computers? What was your first experience with computers?

Ganatra: My first experience was-- I think it was fourth or fifth grade. It was an Apple II. My first experience was actually walking into a classroom and seeing a-- an Apple II with the green CRT display and not really too much impressive on the screen other than text flying by, but it was in the classroom and it was there for us to learn presumably. And so we just had a chance to-- we used it for some-- a little bit more mundane school projects and things like that but there were also what I recall were more text-based games at that time. I think there was a game called Trek or maybe it was Star Trek. I don't remember exactly but then shortly later-- shortly after that Zork came out and of course there were a couple of graphical games that were-- that I became aware of. They were probably available before that but I had never seen them and that was just kind of a different world. All of a sudden this thing started to become a little bit more interesting.

Hsu: So this was the early '80s?

Ganatra: Yeah, early '80s, maybe 1980, somewhere around that time.

Hsu: And when did you first start to program?

Ganatra: I think it might have been a year or two after that. One of the things that we were learning in addition to using this thing for going through-- typing tutorials and math lessons and things like that was BASIC programming and how crazy it seems now in hindsight that there was a programming language and a whole-- a BASIC interpreter built in as part of the computer that shipped right out of the box but it was-- but Applesoft BASIC was there. I think I played a little bit with Integer BASIC but really most of my documentation that I had and people who I knew had all sort of moved on to Applesoft so that was more where my experience was and it was-- obviously the first couple programs were like anybody's programs, was things like "print your name" and then-- on line ten "print your name" and then line twenty is "go to ten," that type of thing, right, nothing super impressive, but I don't recall specifically that that was the very first program that I ever wrote but I- I'm sure it was. It was something equally impressive or unimpressive. Yeah.

Hsu: So then did you continue to be into computers through the rest of your childhood and teenage years?

Ganatra: I did. So I became more and more interested in it and I was-- and some friends of mine were interested too and we were all kind of-- in addition to frankly pirating programs at the time, which we did a lot of, and most of it was just pirating games that had already been cracked. There were masters out there who understood computing far better than me and my little crew did who were actually doing the cracking and would actually come out with a floppy that you could duplicate, but we exchanged those types of things too so software was flying around on floppies, as many as we could afford, but in addition to that there was-- I continued with BASIC. I worked with a friend--I mean I had mostly just little pet projects here and there, little things I wrote to help with schoolwork, but my father was-- had started purchasing and managing real estate and so one of the earliest programs that a friend of mine mostly wrote honestly and I wrote just tiny, little bits here and there was this program that helped him manage the expenses that he had and the rent coming in and what units were open and things like that. And it was just-- it was an amazing thing to be able to take ideas or hear my dad talk about what he wanted and then sit down and oh, well, we can make this-- make-- we can track all of that in this area and we can show that on the display. Again it was all text and everything but it was sort of-- it was an experience that went-- it was more than just "print," "go to ten" or do an assignment for a classroom. It was really sort of the first time that I had seen software written by-- a little bit by me, mostly by this friend who was far smarter than me and actually seeing it deployed and in use in addition to the-- to programs that I had been working on. And so that was something that was very-- it was very I want to say empowering but it's-- it wasn't-- I mean that- that's a pretty strong word but it felt like "wow." I mean this is just something that we can make this thing do what we want it to do and here is an actual demonstration of that happening and what that really looks like. So it was very-- to me that felt very influential that we can do anything on this thing. We just have to come up with something cool to do and some time and we can work it out.

Hsu: Right. So what grade or what age were you when that occurred?

Ganatra: I think I was probably 12, maybe 13 at that time, around that age, so still like early '80s, '80-- I guess I'm giving away my age-- probably like '82, somewhere around there, '82, '83 maybe, something like that. <laughs>

Hsu: And so this is a program that your father used?

Ganatra: Yep. Yeah. He was using it. He had three or four different properties and so he just wanted to be able to enter tenant information into those and all kinds of things like that. I mean at this point in hindsight if we were doing something like that today we might use a spreadsheet but at the time-- and Visicalc was the spreadsheet; that was the one that was out and available. And it was an amazing program on its own but I think it was early enough for me that I just didn't actually understand how

amazing it was. I mean I could-- I understood the concept of cells and that there were formulas and that they could feed off of other ones and sort of-- you can have these cascading calculations and things like that but really it just didn't click with me how to manage a bunch of properties and how to manage these expenses and income and things like that using just a spreadsheet. It was easier to just sort of talk through it and then just develop something on our own anyway and it was just-- and it wasn't that hard of a-- I mean there wasn't any deep math going into this anyway, right; it was all addition and subtraction and what have you so there really wasn't-- In hindsight you might use a spreadsheet or you'd think well, anybody would just plop that into a spreadsheet, but that's just not what we were thinking at the time and it didn't really match up with what we had available either.

Hsu: Yeah. So your father was a realtor or in real estate or--

Ganatra: He was an engineer actually. He was a petroleum engineer but he had been looking to kind of get out as well and so he was-- we had moved back from Saudi Arabia; we actually lived in Saudi Arabia for a couple of years. He worked for a big company there called Aramco and so he was-- he had been in engineering helping to design plants and refineries and machinery around that but then we moved back to California I think '80-- 1980 and then it was a couple years after that that he was sort of-- started branching out and looking at other things to do too.

Hsu: Uh huh. Are your parents first-generation immigrants or--

Ganatra: Yes. Yeah. They were both born in India and they met each other in Canada and then after completing their degrees they both decided to move to California initially.

Hsu: Oh, okay, right, a very typical story, meeting at graduate school or something like that--

Ganatra: Well, my father he had a couple of different engineering degrees <laughter> but my-- I think he may have changed his mind halfway through or decided to do different things where my mother she was an RN and so she was just going to school to get her registered nurse license so yeah-- so it was after that that we-- I think I was two years old-- I was born in Vancouver. They met in the Vancouver area and I think I was two years old or two and a half when we moved here or shortly after-- I have a younger sister; she was two years younger than me and it was shortly after she was born that we moved to California.

Hsu: Oh, okay, and specifically to the Bay Area?

Ganatra: We moved actually to L.A.-- to Los Angeles. I guess in talking to them now the reasoning was that there was still a lot of petroleum-related work in that area. My father before marrying he did live in the

Bay Area and worked at a couple of defense contractors up here but then I guess moved to Canada, got yet another degree at UBC up there and then moved back down to California again.

Hsu: Okay. So then you did your undergrad at UC Santa Cruz--

Ganatra: Yes.

Hsu: --but you weren't initially in computer science.

Ganatra: Right. So I think around 14 or 15 years old-- so I had started-- so I had-- BASIC was kind of well understood. I had started moving into doing some basic assembly language programming as well and just sort of plunking through and understanding what these mnemonics were and that they actually weren't the real machine language but really what was the machine language and understanding basic concepts like registers and a stack and parameter passing and things like that. And so-- but it was maybe a year or two after that that I just sort of burnt out on computers. It had been so much a part of kind of my later childhood and early teen years that I sort of lost interest in it and kind of-- it sort of felt like-- by the time I was looking at colleges and trying to figure out what to do with my life it felt like it was a childhood hobby; it felt like kind of a pursuit that was just kind of a "hobbyish" little thing that you could tinker around with but really once I grow up I need to figure out what else I need to do because I can't just play on computers all day. And so for a couple of years there I had settled on economics and sort of thought that the-- especially the-- sort of the psychological aspects around economics and just what motivates people and what drives them to act rationally sometimes and-- but obviously not act rationally other times. And it just seemed like this whole new world and whole-- this whole-- rather than just having these very specific mechanical input-- mechanical devices that respond to inputs predictably and get outputs in a predictable way it felt less mundane than that. It felt like on the economic side there really was no-- yes, you can build these models to try to understand the outcomes of certain events and things like that but ultimately humans are just humans and they are going to do-- they're going to do things that you just-- you can't possibly study enough to know and predict what they're going to do, and that seemed very appealing to me, just sort of that uncertainty.

Hsu: Right. So you were into behavioral economics it sounds like.

Ganatra: I suppose. I never really thought about it that hard. That was certainly the most interesting aspect of it to me was sort of the behavior side even though I couldn't really-- I've never actually said "behavioral economics" until just now, right, but certainly that was the most interesting aspect and really just sort of the idea that not everything has a correct answer or not everything can be predicted based on identical inputs. It just felt like that that was-- there was a lot more to learn in that area.

Hsu: Right, so why people act irrationally sometimes.

Ganatra: Yeah, right, why they act irrationally or maybe it's not-- maybe it's seemingly irrational but are there-- are they acting in a rational-- actually acting in a rational way and not-- but it's just not apparent to somebody who's observing it and just understanding a little bit more of the context. I mean it really is more yeah, the behavior and psychological and even sort of sociology which is-- The gray areas started to really appeal to me a little bit more and not-- we don't always have the right answers but how can we get close even if we- we're dealing with this wildly unpredictable brain-- human brain. So yeah. So that really started to become more interesting and so I just-- completely just put computers aside, didn't even consider anything sort of computer or math related or anything like that. It was all just sort of-- it was either economics or there was going to be politics or it was going to be something else along those lines, and in looking back I wasn't even-- frankly I'm not even sure what I would have done with a degree in any of those things but I figured that was-- at least these things-- these disciplines led to-- had natural paths into the career world and I'll figure that out after I'm done learning all the interesting stuff. Maybe that's just how high-schoolers think or-- so it wasn't really well-thought-out plans or anything so-- yep.

Hsu: So how did you make it back to computers?

Ganatra: Eventually--so I took a few economics courses and realized that there was a lot more that I was going to have to learn that wasn't as appealing to me as sort of the behavioral side and maybe it was just wandering in as a freshman and a sophomore kind of-- not really-- but pretty quickly I figured out that I don't want to pursue economics. After going to school and taking a couple of courses, it became clear to me that I don't want to do this, I don't know what I want to do, so I may as well take a couple courses here and there and maybe the political side or maybe other things. Eventually, I sort of settled on-- I did settle on politics and I had been working my way through sort of lower-division courses and some of the upper division when it came time to do an elective. I had some electives to fill in as well and one of them was a-- gosh, I wish I remembered what the-- there was a computer class that I ended up taking and it was just sort of an easy elective like "Oh, well, I can just knock this out, it won't be a problem," but very quickly what I realized was this thing that-- so politics was sort of the main focus and then there were just these little classes off to the side that you kind of do to take care of to fulfill your requirements. As soon as I started taking this computing-- this computer class again I sort of-- I realized how <laughs> comfortable the world of predictable inputs and outputs are and how nice it is to not have to-- how nice it is to actually have an understanding of whether something is right or not. Now obviously that- that's-- it's more complicated than that but at the very least you can create things. After having worked for a couple of years in an area where everybody has a valid opinion and everybody has something to say that could contribute and-- I found myself yearning for, well, what is right and what is wrong. And sometimes there are-- two plus two is not five; it's-- <laughs> two plus two is four and there's something very comforting in just being able to hold those facts and then be able to build on them to actually tackle uncertainty in other areas but tackling uncertainty based on those concrete facts that you have and therefore building on it and therefore having something more-- a more complex system that is maybe not a hundred percent provably correct but it's predictable enough and it's-- and in practice it works well enough that it is doing the correct thing based on specifications and things like that. That all started to become-- I sort of realized that that was something that I had been-- missed quite a bit and so <laughs> jumping back into computing it was sort of a oh, my God, why did I ever leave this to begin with? It felt very comfortable, it

felt very natural, and it was easy; it felt like it was something that I could do and pick up on-- pick up where I left off relatively quickly and do well and it was sort of-- it started just consuming all of my thoughts again like it had earlier in life.

Hsu: Right. So what jobs did you apply to after graduating?

Ganatra: I applied to a couple of different jobs.

Hsu: So you did change your major to computer science--

Ganatra: Eventually yes, I did. I think a quarter or two later I changed to computer science and I think I had a little less than two years to sort of-- to complete all of the CS requirements. My electives were largely filled and that-- there were one or two more that I had to but really it was get through all my lower division, get through all the upper division, and then a little bit of prep work for graduation and then I was on to the-- on to yes, finding a job. I think I had-- I applied to a couple of positions in the Santa Cruz area but I was pretty aware that-- I mean Santa Cruz-- this may be true of other college towns but Santa Cruz in particular, it's such a beautiful place to live. I mean you could live your whole life there and just not want to go anywhere else, it's just gorgeous, but the downside of that is that there are a lot of-- what I perceived were a lot of people who once they graduated college they really-- they didn't want to actually go out and pursue a career somewhere or do anything like that as much as they wanted to just stay in Santa Cruz. So you have all these college grads who are in the Santa Cruz area who work at coffee shops or things like that and it's kind of well, couldn't you be doing something more productive? I mean that was just how I was thinking at the time was that well, that seems like an awful waste of whatever degree you got that you-- now you're working at a coffee shop in this-- you could have been working in a coffee shop three years ago and you wouldn't-- and you'd be right where you are now. Anyway, that seemed to be-- so I applied for a couple positions I think at-- there was a company called Amdahl that was still pretty big at the time. They make mainframes and mini-computers I think they called them at the time, a Santa Cruz Operation. I--

Hsu: Oh, yeah.

Ganatra: Yeah, SCO <laughs> that-- they shipped I think Xenix, a variant of UNIX on Intel, and one more-- I don't think I applied to Borland, but anyway I applied to these positions and failed spectacularly in-- to some degree or another to pick up-- to get a job. Another one was-- there was a company in-- I think it was right here in Mountain View called Quorum, Q-U-R-O-- Q-U-O-R-U-M, Quorum, and they-- what they made was a Mac emulator that ran on UNIX systems and so really if you wanted to run Excel on your fancy HP workstation or your DEC Alpha or whatever the heck you would go to Quorum and get their compatibility layer and then run Mac operating system, position. Anyway, I mentioned this-- and I didn't get the job there either so-- <laughs> there were a number-- it was a handful of positions that I was

applying to. Ultimately, I worked at a contract house called Oxford that I-- somehow I had learned that they had a pretty close relationship with Apple and that they placed a lot of people in Apple for contract positions and so that was where I started was at Oxford as a contractor into the developer technical support group at Apple.

Hsu: Oh, okay. Had you been since childhood a user of Apple computers or had you owned PCs?

Ganatra: We were largely an Apple house. My first computer was an Apple IIe and I think the reasoning at that time was well, I-- I mean it was super expensive compared to Ataris and Commodores but my parents were willing to sort of kick in the extra money to get the Apple computer because it matched what we had in school and they saw a real benefit to having the same computer at school as at home and so I was lucky enough to have Apple products at home. When the Mac came out we did not go out and buy a Mac. I mean I thought it was just too expensive; it was not-- a high-schooler, I'm not going to go and buy a Mac especially if I have this Apple II here that does some stuff. It doesn't do what a Mac does obviously but it's a computer and it worked well enough so-- but yeah, we were largely an Apple house up until then. Yep.

Hsu: And so had you had any experience with a Mac before you got that first contracting job?

Ganatra: Not a lot of experience. I mean for me the first time I saw the Mac it was after I had had a little bit of experience already at programming on an Apple II and there was this big-- anybody who's done the-- is-- remembers Apple II programming there was this big switch that you had to do within a BASIC program to kind of put yourself into high-res mode and now-- once you're in high-res mode now all of the operations that you see in BASIC are all about graphics and all about wiping the screen or paint this way or plot from this point to that point or draw this point here or what have you and so when I first-- So that was my understanding of graphical programming was kind of what I had seen you could do in BASIC and so it just-- it blew my mind the first time I saw a Macintosh because the first thing is well, there is no console behind this thing, right. The first moment the screen lights up it's a graphical display and you see the little Macintosh in there and you don't go into-- it's not you-- like you start up in console mode and then put the thing in graphical mode and then "boom," now you're-- now you can use this thing like that. No, this was just a graphical computer; to me it felt like just through and through this was a graphical system. And well, coming from Apple-- or from the Apple II it felt like well, you're in this whole graphical world but yet everything that I'm used to tells me that that's just one half of computing. There's the graphical side but this whole thing is all graphics so what about all of the-- what about the other stuff; how do you handle doing these other things. And the graphical world felt so limiting anyway that my God, how do you draw a window or how do you even track the mouse around the screen. Just these very simple questions were just-- each one of them seemed like rocket science to me and so my-- really my only experience before I worked at Apple other than just playing with a Macintosh was reading Inside Mac. I had worked in the computing staff-- in the computing center at UC Santa Cruz and we had Inside Mac volumes one through six on a shelf somewhere and it was just a-- okay, how do I draw a window. Well, it

was that kind of thing like how do I make a menu that can-- where you have a File and Edit and Open/Close and-- all of those things, how the hell does that even-- do I have to go in and plot the menus and then-- and plot the little apple in the corner if I'm making a program or if that's taken care of by someone else how does that all work. I mean there was just this whole world of-- it felt like understanding a computing system that was entirely different than anything that I had played with before and so it was just this big mystery. And so part of it was just reading through Inside Mac and it was kind of well, okay, these are all [Macintosh] Toolbox calls and I guess there is this thing called a Toolbox that it has and how the heck does that all work and it was very-- it was a very high-level understanding. Basically, when I went to-- when I started in DTS [Developer Tech Support] I had a degree in computer science, I understood UNIX programming, I understood Apple II programming, and I had done Pascal and some 8086 assembly and little things here and there but I had never done anything like graphical programming or Macintosh programming. And so that was just-- in a way I mean I was so thankful-- I'm so lucky to have had that job though because I was paid to learn, learn all this <snaps> as quick as I could because developers are asking questions and they're just-- they're coming in and developers aren't always the most cheerful crowd and you better have a-- if you're going to respond with an answer it better be the correct answer and-- because these guys are paying a lot of money to be part of this program and they expect a high-quality answer. So on the one hand it was a little terrifying jumping in and I have no idea how to respond to these questions about this Toolbox that I just heard of last year or the year before, but Apple I mean to their credit-- I mean I had so many great mentors in DTS and later on in my career I was lucky enough to work-- to continue working with some of these people who as we sort of moved out of engineer-- out of Developer Tech Support and into engineering in various roles so the mentors and then just my own homework and just getting in there and answering questions. I mean there's some saying about how you don't really, truly understand something until you can explain it to someone else, right, so I think that that was a big benefit was not just understanding okay, well, if I use these Color QuickDraw calls but I'm on this type of a system this kind of thing is going to happen. You can have sort of a muddled understanding or an understanding of the basics and fill out the details later but as soon as you're in front of a computer and you have to type an answer out to a developer and you know that they probably know more about how this Mac works than you do but you understand this little detail a little bit better that-- it's a lot of pressure to get the answers right and make sure that you have all of your details worked out too so-- yeah, so it was-- DTS was a great learning experience.

Hsu: Is that like a—I mean--we both know each other from Apple. I remember DTS and I remember working with some of those guys. Is DTS a unique kind of an organization in the industry or do other companies like Microsoft have something like DTS?

Ganatra: I think it's unique today. I think DTS-- at the time there was no Stack Overflow. There was no Internet and so all the information you had available to you was in these manuals and was in this quarterly CD that Apple shipped to you and the occasional magazines that came out about Mac development. So I think that-- I believe at that time it was not a unique thing. I mean I think a lot of companies that had their own platforms did have their own developer tech support organizations. I mean Microsoft was always kind of the envy of-- in my mind was the envy-- well, was in an envious position because there was really this great understanding that developers are what make the platform and your

platform will be successful when your developers are, where[as] Apple sort of had this-- I mean there was obviously an understanding by the leadership in DTS but beyond the leadership in DTS it always sort of felt like well, yeah, we do need developers but is there less that we can do? <laughs> Can we do less to-- and still have them come along. We're spending an awful lot of money paying these engineers to answer e-mails or things like that I mean and there was-- and to be fair there were other things at play too where the more accessible you are to developers and the more-- the quicker you answer questions the lower the quality of the questions that come in, right. So in other words-- and we noticed this all the time where we always strived to have an answer out to a developer I think it was within 24 hours or 12 hours or something like that, I think it was 24 hours, but the problem was that if you got a question, somebody says, "When I make this Toolbox call it returns X and I want it to return Y. How do you get it to return Y?" if you respond to them in a half an hour then the next question you get is going to be even lower quality. In other words, the more quickly you respond to these questions you've now given developers less incentive to go and look things up in a manual themselves because they know-- after a while they'll come to rely on the fact that "Well, I'm going to get an answer in an hour or two, I'm about to go pick up my kids from soccer or whatever, I'm just going to ask this question to DTS and I'll have the answer waiting for me when I get back" as opposed to "Well, I'm going to go pick up this manual and go flip through it for a half an hour and figure out the answer on my own." So it's interesting. There were these-- you couldn't-- <laughs> over the couple of years that I was there I sort of came to realize that you can't answer questions too quickly or you're answering just low-quality questions anyway and it's sort of "Well, why am I even here if you could just look in the-- I'm just a replacement for the manual instead of being a professional who can go speak to engineering or look at the source code or try things out on a couple of different systems?" So DTS was really good at what it did but to me it always felt like Microsoft and these other companies were probably a little bit better. So I don't think it was super unique at that time but I do believe that DTS even now is very different than what it was back then. Yeah.

Hsu: At what point did you move from being a contractor to being an employee?

Ganatra: I think it was within six months; I think it was-- my six-month contract was almost over and there happened to be some new positions that opened up, full-time positions within DTS as well so I applied for one of those. And I mean I was lucky that I had already been working with the team for six months prior and so it was pretty-- I believe it was a relatively easy transition to full time.

Hsu: Yeah, and that was 1990--

Ganatra: That was 1993.

Hsu: Okay. And you were with DTS for another how many years?

Ganatra: Two more years after that-- a little under two years. I think it was the beginning of ’95 that I moved over to system software.

Hsu: Okay, and what prompted that move?

Ganatra: I had a couple of friends who had moved into engineering. I mean it was always sort of dream really was to work--as soon as I became aware that well, this is the team that develops the Mac operating system and I had gone from just thinking that everything about the Mac was just rocket science to having a little bit more of an understanding and a little bit-- just sort of scratching away at how these things are implemented, it just-- I think engineers-- maybe certain-- only some engineers but-- I’m not sure but engineers have a desire to create at some point and so it was really a matter of “Well, okay, I’ve seen how these things have been developed up until now. I want to take a shot at developing some of these new technologies as well and helping developers in creating a better platform as well.”

Hsu: So you joined System Software in 1995.

Ganatra: Yes.

Hsu: So that was the System 7 team-- System 7.5-ish?

Ganatra: Yeah, it was-- so I believe it was either by then or maybe shortly after that that the team itself was called Continuation Engineering and so it was-- so the understanding was that there is this group of engineers-- the previous effort at creating a modern operating system at Apple had failed by then; that was called Taligent.

Hsu: Copland? Oh, Taligent.

Ganatra: Yeah, Taligent—

Hsu: The previous previous--

Ganatra: The previous previous failed-- yes, there were a number of failed attempts. <laughs>

Hsu: But Taligent had been spun off to a joint venture with IBM. Right?

Ganatra: Right. I believe it was the software side of the Apple-IBM-Motorola alliance, AIM I believe it was known as, and Taligent was the software side, but as you say I believe it was mostly Apple and IBM engineers who were developing the Taligent OS and so that had-- I'm pretty sure by then that that was all done and that that was kind of mothballed, but Copland was under development. Copland development was under way and there were already very early builds of Copland, internally I believe it was called Maxwell, so there were internal builds already of what would later become Maxwell that were coming and so the understanding was that Continuation Engineering-- the job of Continuation Engineering is keeping-- the pipeline of Macs that are still coming out keeping that full and making sure that we can-- we have an OS that works on the newest products that are being developed while we're waiting for this modern operating system to get its grounding and be the new OS for all of these Macs. So really we were-- Continuation Engineering was largely-- if there are really bad problems out there in Mac OS affecting a certain Mac product or it could be all Mac products we need to get a fix out there as quick as we can. And for the most part those early releases were just updates so I think the first update I worked on was System 7.5.3 for example, and so I think there were 7.5.3, 7.5.5 and then only later was-- after Copland-- after it became clear that Copland was not going to be the answer to our need for a modern OS that-- that's when work started to really kind of snowball on the traditional Mac OS side and what was once Continuation Engineering became *the* OS team <laughs> in between Copland sort of dying and before the-- Rhapsody or later OS products run by NeXT sort of took off.

Hsu: Right. So that whole time you were doing—so those were maintenance releases, basically bug-fix releases that you were working on?

Ganatra: Yeah, they were bug-fix releases and also new hardware. Apple was shipping new hardware a couple of times a year every year by then; in fact it may have even been more than that but-- because there were Performas that were going out for different SKUs-- Montgomery Ward had a different Performa-- Macintosh Performa that they shipped and it had a different model number than the ones that Sears was selling and that was different from this other-- In the retail space, I guess they play a lot of games with model numbers and-- or they can and if you have a supplier who's willing to play along, and I guess Apple was at that time, then you can make your job-- on the supplier side, on the-- on Apple's side it became a lot more complicated just knowing that a Performa 467 was the same as the Sears Performa 476 because you're just getting a bug report in and now you have to figure out what the hell you're talking about here as far as the machine goes.

Hsu: I thought that was always pretty crazy.

Ganatra: Yeah. It's like mattresses. It's like "Really are we going to do this?" This is like the mattress industry where no-- you can't find the same model across mattress stores and 'cause of all the shenanigans-- anyway-- so--

Hsu: Yeah. This is always something that I found interesting is that every new Mac hardware release required a little change to the operating system and I just thought that was kind of-- why would you need to do it that way?

Ganatra: Yes. That's a fine question. I mean I think a big part of it is just sort of the origins of software as well, right. I mean I think that especially early on in computing I mean there was really just a focus on the product, right. The product was the hardware and the software and so if you had to do-- if you had to-- if you were already doing all of this work to revise the hardware or if you were doing the big, heavy-lifting work to actually develop a new board or develop a new process or what have you on the hardware side then any miniscule changes that you might have to make on the software side well, why are we even talking about that; of course just go ahead and make them. The product itself was this whole thing and just like we were revising the hardware side of this whole product obviously there are going to be revisions we have to make to the software side. So I think that that's sort of where-- the origins of it but just like any habits you kind of-- you probably keep them longer than they're-- than you should and you sort of forget the original reasons why you had them to begin with. I agree though. I mean it became unmanageable especially after we started doing these deals with these third parties to actually build Macintoshes themselves, right. Now--

Hsu: The clones.

Ganatra: Yeah. The clones were now all of a sudden building these products and I mean the way we had been developing Mac OS was we had to make at least minor changes to every revision of Mac OS to be able to run the newest, latest hardware; what the hell does that mean in a world where there's some third party who's building your hardware now and they're not going to-- you're not going to just give them-- let them make changes willy-nilly just because they want to ship a new product, right. So there were-- I mean I know that those were headaches for the licensing group and just trying to figure that out but I think later on there was this understanding that okay, the product-- that software really does have to kind of have its own standing and sort of be a peer of hardware and part of that is having some control over revisions and being able to roll back or being able to work across multiple hardware types and things like that without having these little niggling dependencies. Yeah.

Hsu: What was the kind of stuff that you were actually working on, the kind of things that you did while you were in Continuation Engineering?

Ganatra: So we would take-- if there were escalations-- so if there were escalations either from a company that bought a lot of Macintoshes and they were unhappy because of this one-- this bug in the Alias Manager, you know <laughs> then that became a high-priority fix for us, or the QuickTime team. QuickTime is the video-playback technology that was originally developed on Macs. So if the QuickTime team wanted to release a new-- they had a new version of video-playback software and they wanted to release it across Macs but they really needed one or two different-- one or two changes on some of these

other releases in order to really make QuickTime work well, then there was nobody there to make those changes to system software other than the-- Continuation Engineering. So largely it started as escalations, really, really bad bugs that we understood we needed to fix as soon as possible, and then supporting other groups within Apple who were-- who themselves were developing technology to work on Mac OS but just making sure that it continued to work well across everything that they wanted to support; that was sort of early-days development. And so the cool part is that at any point you could work on-- it was such a small group of engineers-- I mean I think it was like half a dozen engineers very early on and grew from there but it was such a small group of engineers and-- but yet the responsibility that we had was all of Mac OS so really by necessity it meant that you were diving into different parts of the software stack. And so one day I'd be working on virtual memory or working on the Memory Manager, the memory allocator, and the next day I'm working on QuickDraw cursor management. The next day it's the Alias Manager. The next day it's desktop pictures. It's just this-- the Desktop Pictures Control Panel is this-- has these problems and we need fixes in there. So, it was really-- it was neat, because just like with DTS, every question that came in was an opportunity to learn about that particular area of the Mac Toolbox. Every escalation later when I was in Engineering, every escalation that came in or every big chunk that-- of work that had to be done was in Mac OS, and they were all in different areas, and so, it was an opportunity to learn how to-- what is the best way to come up with a solution for this problem that works on these different Macintoshes as well. And it wasn't always easy, too, because the way the Macintosh OS worked at that time was half of the OS, anywhere from let's say 80 percent of the OS and the Toolbox was all the way down to 0 percent for the later licensing ones, but 80 percent to 0 percent lived in the ROM, was baked into the read-only memory on the Mac OS, and there was a strong desire to-- in order to do things like be able to ship floppies that could boot the system or be able to run on systems that had a very limited amount of memory, we really wanted to use as much of the ROM on any Mac OS system you can. But now-- but what that meant in practice was you have this ROM that's just baked in, it's never going to change, and then on top of it you have this system software that contains images that all-- or, not images, but contains code and data that all lives on the disk that has to work with what's in the ROM. And so, what that means is that the initial boot-up process for any revision of Classic Mac OS, the very first thing that it does is look and see, well, what kind of ROM do we have? What's the version of the ROM? And then, based on that, based on the machine and then inferred from that, you can infer what the ROM is, then certain patches were applied from the disk over the ROM system. So, you have kind of this hybrid of most of the code you execute is in the ROM, but then we also have these disk-based patches that have to run for other things. So, you might have the Memory Manager entirely replaced on disk in the system software, but the Control Manager, the thing where you draw little widgets, is all just in ROM, and you just kind of have this mix. And that was-- it made development very complicated. It meant that-- we would joke that it takes a couple of hours to figure out where a bug exists, but then it takes a week and a half to figure out how to roll out that fix across all of the products you want to, because if you want to take that one fix everywhere, depending-- if it's on disk on a certain set of machines, okay, that's fine. You're just loading from disk anyway for those. Well, but now, on these other older machines, that part is in the ROM <laughs>, and how do you just patch-- do you just patch that particular call to make it work? Do you replace the whole subsystem. All these questions start to come about, and it's all because there is this hybrid environment for development. So, it's interesting how-- but you do that for long enough and what amounts to wading through mud. I mean, it really did. It was like wading through mud, right? Like trying to actually get these fixes into software releases. But after you do that for a while, you

don't feel like you're wading through mud anymore, right? You're the fastest mud wader you can-- that you know. You're faster than all your buddies. Look at them. They're all back behind me. I'm an awesome mud wader, you know? Who's going to be faster than me, right? But all the while, what you sort of forget is that you're wading through mud the whole time. Wouldn't it be nice if you were on solid ground? How effortless and easy would that be instead?

Hsu: That also reminds me of-- I remember there was a-- the Mac OS at that point in time was also a hybrid-- there was still some 68K code and-- but you're trying to move to PowerPC native code. How much of-- so, was it-- that all the ROM code was 68K, and the stuff on disk was PowerPC, or what was that breakdown?

Ganatra: Oh, no. There was native code that lived in the ROM as well, and that did have to be patched as well. It was-- from what I recall, it was-- most of it was low-level drivers that were written in PowerPC that were executed out of ROM. But you're right. I mean, that was another sort of dimension on this whole problem of, okay, I have this fix <laughs>. I know it needs to work on PowerPCs, and here's the changes in-- maybe in C code, right, to actually build this new native shared library that's going to run on these PowerPC systems. But I also have to make that work on this other system where that native library's actually coming from the ROM itself, so that we would have these PowerPC code fragments, they were called, that were actually baked into the ROM, too. So, we actually did have to create a whole other mechanism for overriding these code fragments that were in-- on the PowerPC ROM. So, really kind of <laughs>-- it meant that just to understand how to deploy a bug fix, you really had to understand how the system was implemented for that particular machine, right? And so, you can imagine as soon as you have a dozen different machines, and each one has maybe a slightly different-- or a way of running code, each-- I mean, each machine that comes out is effectively a snapshot of system development, of system software development, right? And so, because of that, those snapshots are all different, and because the snapshots are different, it means that when you want to go and update that particular system or that particular Macintosh, it means you have to come up with a unique way of patching that system that was a snapshot taken at that time. So, really, just like the ROMs are all different, the ways that you update the ROMs are different across machines, too. So, it became very complicated very quickly.

Hsu: Wow <laughs>.

Ganatra: Yeah, kind of a mess, but I think-- but-- and I think in Apple Engineering's defense at the time, when all of these things were being created, there was also this feeling that, okay, this is all just-- we just have to get this working well enough until Taligent comes along and-- we don't have to worry about this ROM in RAM stuff anymore. We're going to have a real virtual memory system and things like that, and then, a couple of years later, oh, well, okay. That didn't work. Well, we just have to keep this limping along until Copland comes along and shifts, and then we can ditch all of this stuff, right?

Hsu: Right.

Ganatra: So, I think all the while, it was sort of an understanding that, well, we're-- we really shouldn't spend too much time coming up with a long-- a much better solution for this problem long-term, because long-term, this whole problem is going to go away. But really what it meant was because we never-- we didn't actually manage to ship a modern OS until the next decade, right? It meant that we had these festering problems, and they would just kind of fester and we would just-- we just had to manage them the best we could until the new thing eventually did come along.

Hsu: Yeah. And so, before we get into that, I do remember that later on, I think it was-- maybe it was after the NeXT acquisition, there was a transition to, quote, "New World Architecture" that-- well, that had a different arrangement, like the ROM was now in RAM.

Ganatra: Yep.

Hsu: So, what was that transition like?

Ganatra: So, yeah. So, New World-- so, the problem was that New World--

Hsu: --or was that something done for the PowerPC consortium?

Ganatra: Yeah. But that was New World--

Hsu: --the CHRP [pronounces "chirp"] thing.

Ganatra: Yes. Exactly. That was done for licensees primarily, and that was-- and part of that did come out of this understanding that third party Mac developers aren't going to be able to develop a Mac the same way that we did within Apple, because we had the ROMs and we could bake them into anything. And so, yeah. So, New World was really the software platform for CHRP, the-- I forget what it was called. Something Hardware Reference Platform.

Hsu: Common, I think. Yeah <laughs>.

Ganatra: Common Hardware Reference Platform. There we go. That's what it was. Yeah. <laughs> And so, yeah, the idea there was that we're not going to have this baked-in RA-- this ROM that's baked into machines anymore, because then we would have to distribute ROM chips to licensees, and for whatever reason, that was seen as undesirable. But instead, what we're going to do is we're going to have a ROM that is loaded from the system disk itself, and in order to make it so that we can keep the rest of our system largely the same as we have-- instead of-- I mean, you could just say, well, why even

have a ROM that's resident in RAM? That doesn't even make any sense. It's just bits that you load off the disk, and now they're resident in RAM. Why are you even conflating this with the idea of ROM? The reason is really just because the whole rest of the system had been developed in such a way that the ROM was the significant piece, and so, it was much easier to actually create this-- what seems to be this funny architecture where you have this ROM in RAM and make the whole rest of the Mac operating system work the way it used to, as opposed to just what would amount to boiling the ocean. If you just got rid of this-- the idea of a ROM entirely, then now you have these cascading changes going through all of Mac OS. Well, and by the way, Mac OS still has to work on those systems that do have a real ROM, too. So, it's much-- so that-- that was the most logical, best way to actually tackle this problem and still make it so that third parties could develop Macs without a ROM chip in them.

Hsu: Right. And so, was-- the ROM in RAM was just-- so, that was considered its own hardware platform--

Ganatra: --yes--

Hsu: --separate from an individual Mac? It was like a virtual Mac kind of a thing?

Ganatra: Right.

Hsu: Okay.

Ganatra: Right, and they had-- and the ROM in RAM project, or the New World project, the awesome thing was that they could just build a new ROM anytime they wanted, right? And they didn't' have to worry about how to take changes for this particular configuration. They were just all-- they were all loaded from the system disk, anyway, like a reasonable, modern operating system should be, right?

<laughter>

Ganatra: So, yeah. They were-- so, they were the-- that was the first system to actually make that leap to having just an entirely loaded from disk system that worked on the old Macintosh architecture.

Hsu: Right. And so, that had been developed for the clone makers, but Apple eventually used that for its own machines later on.

Ganatra: Did it? I guess it did. Yeah, yeah, yeah, it did for the iMacs.

Hsu: Yeah, the iMacs and then Blue and White G3s and all subsequent PowerPC Macs.

Ganatra: Yeah. Exactly. Exactly. I mean, that was-- yeah, you're right. That was-- New World became sort of the new ROM for all of these new systems, and I'm pretty sure it started with the original iMac itself. That was a New World product, the first New World product. Yep. Yeah. Yeah. Thank goodness then it was-- I mean, by then, the problem was that I think by then I had already started sort of working on what would become OS X [pronounces "Ten"], and so, I was less and less involved with classic Mac OS by then. But, yeah, the couple times that I did have to go back and work on some things, it was very liberating to think, oh, well, I just have to check in a fix here, and it's just automatically going to be picked up by iMacs and things-- the way you would hope things would work. Yeah. So, that happened shortly after my time.

Hsu: At the point where you were still in classic System Software, who was running-- who was the executive in charge of System Software?

Ganatra: It was Steve Glass. I'm not sure if you remember that name.

Hsu: I-- but it sounds familiar. I do think I-- I do think I remember that name.

Ganatra: Yeah. I believe the--

Hsu: --I don't think I've-- I think I never met him, though.

Ganatra: Okay. Yeah, yeah. He-- I believe he reported to Ike Nassi, who was the previous head of software-- Macintosh software at Apple. So, Steve Glass reported to Ike Nassi, and then the Copland side reported up to Ike as well.

Hsu: Oh, so Ike had both the sort of System 7 division and the Copland division both reporting to him.

Ganatra: They both reported to him, yep, and Steve Glass was our VP who reported to Ike. But-- and, obviously, Ike was heads down on Copland and other things, too, I suppose, but-- so, yeah, mostly our point of contact was Steve-- Glass.

Hsu: Okay. And that was before-- and Ike was the head even before Gil Amelio came along or Ellen Hancock came along?

Ganatra: I believe-- see, now, I-- we would have to go back and look. I don't remember exactly. I thought that Ike reported to Ellen, but I'm not really sure about that, and as far as Gil goes, I mean, I think that Ike-- I mean, Ike was there for the Gil Amelio era. I think that he was hired and became the head of software during Gil Amelio's time there.

Hsu: Right. Okay, because I think-- we've actually spoken to Ike, and he had originally been in ATG at Cambridge, and then-- and he work-- and then he came and he worked on some developer tools and at some point he became head of system software. Yeah, but we didn't-- he didn't talk much about those days.

<laughter>

Ganatra: No.

<laughter>

Hsu: We'll have to ask him about it some other time.

Ganatra: Right. Right. I mean, I'd be really curious about MkLinux and what his--

Hsu: --he did--

Ganatra: --involvement was with--

Hsu: --he was-- he did mention MkLinux. He wanted to talk more about MkLinux than Copland. He didn't say anything about Copland.

Ganatra: <laughs> Yeah. That's interesting <laughs>. Yeah, that was-- I've spoken to other engineers and engineering managers from that time, and that's-- that seems consistent with what--

<laughter>

Ganatra: --with their interactions with Ike as well was. He was far more interested in MkLinux, which was seen as this side project that wasn't really the future of the company, as opposed to the big looming project that has problems and is maybe not going to ship on time or what have you but is the future of the

company. So, anyway, I don't know if there's any-- how much truth there is to that. This is just-- it's all second hand from what I've heard from other people.

Hsu: Oh, okay. So, you were getting that sense from other people, that he was probably more into this--

Ganatra: --<laughs> he was more into MkLinux than Copland, even though he was in charge of the Copland project as well.

Hsu: That is really interesting. It-- what was it like sort of living through that Copland debacle, I guess I could say <laughs>? I mean, you guys were trying to keep things running while they were supposed to be producing the next thing, and then suddenly one day it's-- they're gone.

Ganatra: Yeah. It was--

Hsu: --I mean, did you see the train wreck sort of slowly approaching, or...

Ganatra: Yeah. I did. I mean, I had good friends on the Integration team, the Software Integration team. And so, they were this group of engineers who really understood-- who really had a good understanding of large swaths of the OS and how they were supposed to interact. And so, the Integrate-- but sadly, I think a lot of the work of integrating these components fell on the Integration team as well. In other words, groups would sort of kick their subsystems over the fence, like the window-- like the High-Level ToolBox team or the QuickDraw or the graphics team or the file system team or the kernel. They would just sort of develop their little pieces in a vacuum, maybe run some rudimentary tests or what have you, and then check it all in and let the Integration team actually figure out how this thing was supposed to work with all of the other brand new components that were developed similarly, right? So, yeah, as you can imagine, the Integration team was very busy <laughs>. Integration was this huge cost at the time based on how development was being done, and even though they were some of the most capable engineers at Apple, in my opinion, they-- I mean, they were just overloaded and they were doing this work that really should've been handled by the specific engineering teams rather than lumping them all on Integration. And so, yeah, so, I could see at the time. I mean, there were problems where a couple of weeks before a big milestone, engineering teams are throwing their pieces over the fence, and the Integration team is trying to cobble them all together into a release, and it's just day after day of waiting for this team to fix this problem, waiting for that team to fix that problem. Now, I mean, all integration teams in a sense do that, and that's sort of what the work looks like for-- you could describe a well-run OS project in the exact same way. The integration team is sort of careening from crisis to crisis and putting things together and coming out with a release as best as they can. But, really, the problem was that even when a lot of these little pieces were fixed, just the overall system just did-- it did not work very well at all, and it just-- it was this brand new, very wobbly, barely working OS from the very first time that I saw it until what was supposed to be a release that went out to developers until I saw it at that stage as well. It just

never-- I mean, there were some really cool parts about it, and then talking to the teams, there was some really cool technology in those respective pieces, but it just never worked that well together. And that was something that sort of stuck with me later was you really can-- you are allowed to kind of look at a piece of software and say, well, it doesn't-- no amount of tweaks-- this thing is not just a handful of tweaks away from being able to ship. This thing is a couple of years of doing big work to actually get this thing to ship and just sort of being able to see the difference and understand based on how fast changes are coming in what trajectory you're on as far as going from something that's wobbly and unstable to something that you're proud to ship to customers. It just never-- it just sort of went on this flat line of just being this wobbly, unstable thing, and it was just wobbly and unstable in different ways in each release. And so, there wasn't-- it just was not converging. Things just were not ever moving towards something that looked like it was shippable.

Hsu: Right. Was there any particular reason, do you think, that was the case? Was it a-- it was an architecture or design or the technology or the way the organizations were run?

Ganatra: I mean, for-- it's hard, I mean, to really pin down any one. But, I mean, for me it comes back to management and comes back to prioritizing-- having something that works well, and that was something that I-- that was supported later when-- after the NeXT acquisition and just sort of seeing the difference between how things were run before and how things were run after. But for-- in my mind, it was more-- it was-- I know that there were stories about infighting between this group and that group and there were stories about the kernel team not being happy with the VM team or-- to different anecdotes here or there, but, to me, anyway, I mean, just in my opinion, that always comes down to management. If you have management that's participating in infighting or not doing anything to bring teams together and actually look for solutions, I mean, that-- yes, the teams are at fault for participating in this, but, really, management really needs to come together and come up with the priorities and sort of stop the nonsense as quick as they can. I mean, you know that the job of shipping an operating system is-- it's a huge undertaking, and if your energy isn't all devoted to that one particular task, if instead you're trying to screw over that guy or cover your ass from what this person is saying, it's just-- you're not going to have enough time and you're not going to have enough energy to ship a great OS. And so, you won't <laughs>. You'll either ship a shitty OS or you'll never ship at all and you'll be off-- projects get canceled. You'll be off doing something else. So, really, it just felt like there were a lot of distractions and a lot of-- and management just was not coming in and knocking heads and getting everybody on the same page like it would later.

Hsu: Right. So, let's walk through that period from the cancellation, the decision to cancel Copland, to the NeXT acquisition. What was that like from your-- where-- sitting where you were?

Ganatra: I-- this-- it's tough, because I don't specifically remember-- it's not like a presidential assassination or 9/11 where, "Where were you the day that Copland was cancelled?" I don't have any memories like that of when Copland was canceled. I have more memories around when Apple decided

to acquire NeXT than I do-- and the time-- and what that was like then when Copland ultimately fizzled out. I mean, it-- to me it-- and maybe this is a problem, because to me it felt like it was fizzling for a long time-- for a good long time before it was actually finally killed. That may be the problem that I'm running into here, but I don't remember-- I mean, what-- there-- I don't remember there being a time between Copland dying and us saying, "Oh, my gosh. What are we going to do?" to "Okay, now we've made this NeXT acquisition. Everything is going to-- things are going to get better or at least we have a plan." I don't recall there being a window like that, and so, it almost-- so, I don't know if that's just because-- again, it felt like it was fizzling the whole time or maybe Copland was officially killed. It's even possible, and I just don't know timeframe wise-- it-- you're probably better off asking somebody who worked specifically on Copland when it was killed, because I don't remember there being this window where I was thinking we don't have a plan and we don't have-- we just don't have a plan. It felt like we had that bad plan that was kind of fizzling out right up until we made the NeXT acquisition, and then we had a better plan after that, to me, anyway. That's-- I'm-- yeah, I wish I could remember more about that, but there wasn't a party and there wasn't any-- a funeral or anything like-- it was nothing that I recall like that. It was sort of a long, slow death for Copland.

Hsu: So, how did things change for you right after the NeXT acquisition?

Ganatra: Well, the first thing was I was very happy that we did not buy Be, just-- in my own limited tinkering and understanding of the Be operating system, there was a lot of sizzle there, but it didn't feel like there was a lot of steak behind the sizzle. It was unclear to me what were they going to do about international support, and has anybody tried to print something on a Be system yet or-- I mean, there were these little holes here and there that really weren't-- there weren't great answers around, and, okay, there was a toolbox that was multi-threaded, but users aren't multi-threaded, like the way you interact with a computer is all pretty much single-threaded. So, does that really buy you as much as you think it does? There were just these kind of questions around-- every cool thing that Be had, to me it felt like it had a big glaring question attached to it. And so, it was kind of a, well, do we really-- and, by the way, they hadn't ever shipped commercially up until that point either, right? And so-- but their hardware program had been scrapped, and they'd just become a software program at that point, but they still hadn't quite hit 1.0. It's not like people could-- you could go buy Be software and go run it on your BeOS machine at home or whatever, right? I mean, that was still early days for that. So, all of that sort of made me feel like what do we-- do we really want to get in with this type of thing? When I heard about the NeXT acquisition, I felt like okay, at least this is a modern operating system with a graphical interface on top and it has actually shipped. It's been shipping for a-- quite a while at that point, and so-- and anytime you can get to the point where you're shipping, that means that you've tackled the ten thousand questions or ten thousand issues that you're going to have that stand between you and shipping, and that's a really big deal, is addressing all of those things. And those are the things that really come to the forefront when you have actually shipped, and if you haven't done anything to address those issues. And so, the fact that NeXT had actually been shipping for a while and shipped on multiple platforms and I knew what a NeXT box was like, and I'd played with them, and I liked them quite a bit, that all felt much better to me. It felt like, okay, we're actually buying something real here rather than just some demos and potentially smoke and mirrors with some other solution. NeXT felt like a real thing to me. So, that was a big relief, that part was.

Hsu: And then, you mentioned earlier that now there was work to do in classic Mac OS system software that was not just maintenance, but it was now new features and it felt like there was this big shift to actually put out new versions of classic Mac OS.

Ganatra: So, that's a good point, because-- forgive me. I'm trying to put pieces all together in the middle of an interview. I should probably have had it all--

Hsu: --no, that's fine <laughs>

Ganatra: --outlined before coming in here. But, yeah, so, there was-- so, the Continuation Engineering team went from being-- just dealing with escalations and really nasty problems and supporting other teams to all of a sudden now rolling out new features and new functionality.

Hsu: Right. And this is after Avie had taken over as the vice president of software?

Ganatra: Yes. Yeah. This was after Avie had taken over. I know that because <laughs> there were some Mac OS 8 pictures that were sort of shown around, and Avie was in these pictures, even though he wasn't-- I don't remember Avie being too involved at that time. I mean, I think that he had just taken over, and Mac OS 8 had just shipped. So, based on that, it does seem like Copland had fizzled out for quite some time before to the point where there were teams that now all of a sudden they had this technology. They had these new things that they had been developing.

Hsu: But there was a System 7.6.

Ganatra: There was a 7.6 [pronounces Seven Six]. Yes.

Hsu: That was released before or after the NeXT acquisition?

Ganatra: That was before.

Hsu: That was before.

Ganatra: Yeah, that was before. There was 7.6 and 7.6.1. They were both released before. They had-- I don't think they had-- they did not have the new look and feel, like the new Appearance Manager look and feel. They didn't have the new multi-threaded Finder. They-- I mean, I think that it was-- but it was still called a release. It was different form 7.5.5. I think there were a lot of VM changes that went-- virtual

memory changes that went in. There had to be more to it than that. Now, I just don't remember what it was. But yes, you're right. So, around-- so, sometime after 7.6, that's when there was this work-- that was when a lot of teams started coming around and sort of had this understanding that-- so, by then either Copland was dead or there was an understanding that Copland was going to ship so far into the future that they want to ship this stuff sooner. And so--

Hsu: --like the new-- the appearance--

Ganatra: --right. The new appearance--

Hsu: --the new look--

Ganatra: --and the new-- I believe the multi-threaded Finder were the two that were sort of a, well, we have these things, and we want to ship them, but how do we do that? And Copland's going to take way too long. We don't want to wait for Copland. Or maybe Copland was dead by then. This is where I'm-- this is where the timelines start to blur for me for when Copland actually died. But we did-- but I do specifically remember all of a sudden we went from being this Continuation Engineering team that was kind of bolted on the side of the OS group to being the next big ship that was going to leave Apple, as far as the software goes, was going to be hosted out of what was originally the Continuation Engineering team, right? So, that was the team that shipped. 7.6 was largely developed by the Continuation Engineering team, and by the time we got to 8.0, it was the Continuation Engineering team combined with a couple of these groups from-- that were reporting into Copland but wanted to ship before that. And then, by the time we got to 8.5, there was even more of that, right? So, there was a native Control Manager, native Window Manager, native-- more and more of the components that were originally slated to go into Copland, they all started rolling into traditional Mac OS for 8.5 and 8.6 and beyond. So, it was around that timeframe that-- so, I think 8.0 was when Avie started. That was, I believe, his first release. Yeah.

Hsu: And so, what was it like for you working on new features for the first time?

Ganatra: It was <laughs>-- it was great. I mean, it was fun because-- I mean, it was fun and a little terrifying all at the same time, just because there's a certain amount of comfort in knowing that-- when you're-- when you go in and you're a firefighter, the people are very happy once the fire is put out, and then you can go home and you're done. It's sort of-- you don't have to worry about clean-up. You don't have to worry about a beautiful experience. You don't have to worry about all-- first time user impressions and things like that, right? It's sort of a, “I was here to put out the fire. The fire is out. Go, me. I'm out of here,” right? To, all of a sudden, now it's a “okay, well, we've got this new Toolbox, and we've got this new Appearance Manager, but we still need to run-- we still need to make sure that we run across this large-- these swaths of hardware that run as well as we ever have. And what are we going to do about a

boot floppy with the new look and feel, and-- yeah, I mean, it was-- it-- the act of shipping-- and, also, in addition to all of that, we were now working with these teams that were used to developing on-- developing with little regard for sort of constraints, as far as engineering goes. In other words, they didn't have to worry about a fixed heap on Copland, right, for example. So, if they just want to allocate, allocate, allocate, [memory on the heap] they can do that to create the best possible experience and just understand that they have a sophisticated OS under the covers that's going to make everything work pretty well. On classic Mac OS, we've never had that. It was a-- you have to be very, very frugal with all of your resources to provide a great experience, and we were now working with engineers who had never come from that environment.

Hsu: So, they were just allocating memory on the heap just willy nilly and never cleaning it up?

Ganatra: Cleaning it up, but yeah. So, I mean, it wasn't leaks, but it was still using far more memory to create things or maybe if a team was coming up with a way to implement something, they may just go with the first data structure that came to mind and realize that, oh, well. Okay, I just allocated a megabyte array. Yeah, that's not a big deal. I'll just do that. Whereas, on classic Mac OS, it's "holy shit! What's this one megabyte block sitting here right in the middle of our fixed heap?" It was that kind of thing, where it was a "oh, my God! You can't-- yes, I know in computer science you're taught that, well, stacks are infinite, and that's how recursion-- that's why there are no issues with recursion or what have you. But you can't develop on classic Mac OS like that. You never have been able to. You always have to be very frugal with your resources and understand that memory is a fixed resource and that once it-- once you've exhausted it, it's not like you're going to page or anything like that. No, no, no. You're done. Memory is gone. You're not going to be able to allocate anymore. Just those types of things. You hit these very hard limits. Classic Mac OS was very brittle in those ways. If you went writing off the end of a heap block in the-- and you happen to be in the heap where a lot of system resources were allocated, you could screw up something in the underlying system. Or if you exhaust all available memory, you might exhaust it for the whole system. There are all these things where the actions you take could impact far more than just your little component, and if you're used to developing things in a modern system, coming back to something like classic Mac OS can really feel like sticks and stones. And so, some of that was learning, too. It was sort of, hey, I understand you allocate these buffers to maximize your I/O throughput, but you're also-- now this user can't open a new document anymore or things like that, right? So, there was some amount of learning and working with these groups to actually retrofit their technologies and make it so that they do work well on something like Mac OS 8 or 8.5. There-- that was a good amount of the work that was done at that time, but it was exciting, too. I mean, finally it was a—"Oh, my God! I only have to-- if I want to fix a bug in the Dialog Manager, I only need to look in one place? That's fantastic! I don't have to figure out what the Super Mario ROMs did or anything like that." So, yeah, it was exciting but it was also-- the job did change as well.

Hsu: Right. So, then you were part of that group through 8.0, 8.5, 8.6?

Ganatra: 8.5, I believe, was my last release, and then I think 8.6 was when I'd started working more on sort of the-- figuring out what APIs-- I think by then the message around shipping a NeXT-based system that looked like Mac OS, that had been tried or that had been floated to developers-- I think the initial release was called Rhapsody, which was a-- it was sort of the new Appearance Manager look and feel on what was otherwise a NeXT-- an OpenStep release, and that would ship on Macintoshes, on Macintosh hardware. But what it really meant was that nothing-- no traditional Mac software ran on this thing. It was all-- only NeXT software that did run on it, and that was the-- one of the first things that was pitched to developers as, “here is the new Mac OS. Here-- it's the same as last year's NeXTSTEP, and by the way, you have to rewrite all of your software to work in this new world, on this new modern operating system.” And that message fell flat pretty quickly with developers. And so, there was this work that was done to figure out, well, okay, if we are going to-- the big pushback from developers at that time was, you're going to make me throw away my investment. We've been doing-- I've invested all of this engineering effort and development cost in the Mac Toolbox APIs and making something that works really well on a Mac, and now you're telling me I have to throw all of that out and rewrite from scratch using Objective-C and Cocoa APIs or AppKit APIs at that time. And developers just weren't buying it, and especially at that time, Mac market share was going down. Things weren't looking great as far as the future, anyway. There were questions about this acquisition, and developers were in a position where they were just saying, "We're just going to develop for Windows, and if you're telling us that we have to start from scratch for this new platform that now has eight percent market share, or five percent of the PC market share, why would I do that? You tell me." <laughs> And so, very quickly Apple sort of went back and said, "Okay. We need to preserve developers' investment in their source bases until they see the value of actually developing for the modern operating system all on their own. So, what can we do about that?" And so, that's when the work-- the hard work of going through the thousands of Mac Toolbox APIs took place and figuring out which ones are we going to support? Which ones can we get rid of and not piss off a lot of developers? Just sort of classifying the APIs and figuring out what our story was going to be around that.

Hsu: And so, that's what became Carbon?

Ganatra: Correct. Correct. The APIs-- the subset of APIs that we agreed we would carry forward to the new modern operating system was called Carbon, the Carbon APIs. Yep.

Hsu: Where did the name Carbon come from?

Ganatra: It's my understanding it came from Steve.

Hsu: Steve Jobs?

Ganatra: Steve Jobs and that carbon was the-- and the reason was that carbon was the building block for everything. And so, the-- so, the Carbon APIs were going to be the building blocks for this new OS in the

future. Now, technically, that wasn't actually the case, and it's probably a good thing that it wasn't, but I think that at least it conveyed to developers how important this was and how we're taking their feedback seriously, and we're going to take the solution seriously as well.

Hsu: Right. And I remember-- Scott Forstall was big-- very involved in that effort.

Ganatra: Yes.

Hsu: Correct?

Ganatra: Right. Right. He was very, very involved. He was-- I think it was Scott Forstall and my boss' boss, Tim Swihart--

Hsu: --oh, Swihart. Yeah--

Ganatra: --who were both kind of working--

Hsu: --he was also my boss' boss.

Ganatra: Was he really?

Hsu: Yeah <laughs>.

Ganatra: Oh, that's funny <laughs>.

Hsu: But I was in a-- well, I joined in '99 in classic Mac OS on the File System team and I was working for Andrew Poupart, and so, my boss' boss was Swihart.

Ganatra: That's funny. That's funny. Yeah, I love Tim. I still-- yeah, we still talk and visit. I've seen him a couple times over the years, but, yeah, he's a great guy.

Hsu: I think, technically, we both shared, both Swihart and Forstall, was both our bosses' bosses <laughs>.

Ganatra: Yeah. Yeah. It's funny.

Hsu: At different points in time. <laughs> So, Scott was collaborating with Tim on that effort.

Ganatra: Right. Right. Scott-- my first involvement that I had with Scott was around the development of a demo that was going to-- that we were doing for Steve Jobs, and the demo was the-- I don't know if you remember the app ClarisWorks?

Hsu: Yes.

Ganatra: So, ClarisWorks-- we had the code base for ClarisWorks, and it was understood that ClarisWorks is a big, sophisticated, office-style app that was pretty successful on classic Mac OS. And so, the goal was to actually get ClarisWorks ported and running on top of a Rhapsody system, and so, what that meant was the very earliest-- it was sort of the earliest attempt at a Carbon app, what would later be known as a Carbon app, because it was largely a C-- I mean, it was a C code base <coughs>, and it was all programmed to the Macintosh Toolbox. And so, we had this-- we had the beginnings of a Mac Toolbox implementation that the QuickTime team actually had implemented called QTML, the QuickTime Media Layer. Really what that was, was a very small subset of the Mac Toolbox APIs ported to various platforms. So, I think first starting with Windows, but then there was an HP/UX version of QTML. There were-- there was a Linux version under development, and so, the very early work was done by some guys from the QuickTime team <coughs> who took QTML, ported it to Rhapsody, and then we ported ClarisWorks on top of that. And so, it was the first time that we had to prune off large swaths of what wouldn't be demo'd and maybe, you know, code that had to be thrown away or it couldn't be ported easily. But then also, making sure that the, you know, that the code, you know, to control the Menu Manager worked as well as it could, you know, it worked well enough so that ClarisWorks had a-- gave us a convincing demo on top of Rhapsody. So I don't remember how many weeks we worked on that, but it was a-- it was a huge amount of work. But eventually, we ended up with ClarisWorks actually running, it was fairly clunky looking too, because it was using QuickDraw coordinates and QuickDraw drawing on the Rhapsody system. But we actually had it up and running and we were able to sort of go through and create new documents and, you know, modify existing ones and things like that, and sort of demo'd that to Steve Jobs very early on as a proof of concept of the Carbon concept. So that was the first time that I worked with Scott was on that side.

Hsu: And you were still with the classic Mac OS group.

Ganatra: At that time I was still with the classic Mac OS group and so when I was on the classic Mac OS side, it was, you know, the job was going through this, you know, this mountain of APIs and, you know, you're in, you're out, you're in, you're out. You know, here's why you can be out and, you know, things like that.

Hsu: So it was deciding, like, like why would things be thrown out versus included?

Ganatra: So, so we wanted to throw out as much as possible.

Hsu: <laughs>

Ganatra: Right? Because everything that you throw out is just, every API you throw out is an API you don't have to support down the road. Bertrand [Serlet] later, you know, sort of captured that sentiment in a great way by saying that APIs are a liability, you know. They're an asset to your company, but they can also be a liability. The more APIs you have, the more exposure you have to programs that may not work well or make assumptions based on the behavior of those APIs. So ultimately, we wanted to throw out as much as we could, but really, there was-- but we had a lot of good data by then where we had these tools where you could scan an application, you could scan a popular application and look and see what are the APIs that it's actually using. And so from that, you know, you could bring up Microsoft Word and run it through these tools and it would say, "Microsoft Word is using 2000 API calls. Here they all are." You know, and you could even flag by then, like flag APIs that--we really want to get rid of these ones, but we don't know if we can yet. So let's run through the list of popular apps and see, by getting rid of that API in practice, who are we hurting here? So we really wanted to get rid of as much as possible, but, you know, the other thing, too, is if we can find modern replacements or if we can find replacements that are implemented in a way or that where the interface itself is defined in a way that it hides some of the underlying behavior and hides some of the assumptions that you might make about this thing, you know, this program running on what was-- I mean, Mac OS in, by today's definitions, was largely an embedded OS. You know, and by today's standards it's an embedded OS. So really, and we wanted to make the, you know, we wanted to make these programs work on a modern OS. You know, something more modern with, you know, a modern runtime and preemptive multitasking and memory protection and virtual memory and things like that. So if you had these APIs that sort of gave away the fact that you know, you were running on an embedded OS, those would be very, very hard to port over or they'd-- or it'd be awkward to port those over to a modern operating system.

Hsu: Because they expose the underlying implementation and then the developers rely on that implementation, so then you can never change it?

Ganatra: Exactly, exactly. Yes, yes. They expose too much about the underlying implementation or they expose behaviors of the underlying implementation.

Hsu: Right. Or like data structures, things like that.

Ganatra: Right, right. Exactly. Data structures, you know, that we may want to extend down the road. You know, in the old days, data structures were sort of fully defined and fully fleshed out as part of the

definition of an API call. And part of the reason for that was just that, I mean, you know, just from an engineering point of view, you, you know, you really don't want to create multiple calls to get and set things out of a data structure. Really, the more resource friendly way of doing that would be to just say, "Here's the data structure, you can get, and pull, you know, get and set things in there all you want and, you know, here's the definition of them." And, you know, and you save yourself creating a bunch of calls for those things and you know, what back at the, you know, in the mid-eighties would have seemed like an excessive amount of, you know, function calls just to get and set these things that are very, very simple to understand, why would we want to do that? We only have 128K of RAM to run in anyway. So let's just do away with that and just expose the underlying fields and just let the developers do what they want. So fast forward 15 years from, you know, from that attitude, and really, it does become more important to hide the underlying implementation so that you can change it later, you can extend it later. And so, so a lot of work was done to sort of hide the underlying implementation or make it so that there was just a modern equivalent that made it so you didn't have to do these things that you would on an embedded system.

Hsu: Right.

Ganatra: And so a lot of it was just cleaning out the API as much as we could without making developers' jobs that much harder. You know, we didn't-- we really didn't want to make gratuitous changes, but at the same time, there were changes that we needed to make to make it so that we could have a Carbon Window Manager that worked really, really well on CoreGraphics and on Mac OS X [pronounces "Ten"] technologies. So it was sort of this riding this line between giving developers everything they want and, you know, tying our hands for the, you know, for future development work, or, you know, dial it back as much as we can until developers almost scream that it's too much work to do, but now we're set up well to support them in the future, and just sort of trying to figure out where that was the whole time.

Hsu: Right.

Ganatra: Does that make sense?

Hsu: Yeah. So then you also-- so then you also worked on a piece, there was like a little thing called the CarbonLib, the Carbon Library that was dropped onto the classic Mac OS at that point in time. What was-- Why was that part necessary? Like why was it important to have this thing that you stuck onto classic Mac OS?

Ganatra: Right, right. So really, it was a matter of giving a message to developers that made sense. You know, developers were already used to shipping just one piece of software, and as long as they could sort of define up front what Macs it worked on, you know, they were fine and they could make sure that it

worked well across, you know, a limited set of Macs. The thing is that, you know, once you're actually implementing something or once you're developing a program and you want it to work well on this modern operating system, but that modern operating system really doesn't have much of a market yet. You know, it only works on, let's say the highest of high end Mac products that you've shipped out there, but you've got software and you want to ship it to as many people as possible, you really want that software to work on as many platforms or in as many environments as possible, including the old, you know, the old non-modern operating system. Really, the most desirable thing for a developer would be "I'm going to go in, I'm going to use this one toolchain. I'm going to do all of my implementation work. I'm going to make sure it works really well on Mac OS X. And without making any modifications, I want to take that same binary and I just want to put it on a Mac OS 9 system or Mac OS 8.5 system and see it run really well there too. So how do I make it so that I can build one Mac-- Macintosh application and have it run on OS X and on OS 8.6 or OS 9 as well? And so that where the idea for CarbonLib came was, we knew that we were going to have this subset of APIs on Mac OS X because it was the modern OS and we could really, frankly only support this subset without doing a huge amount of extra work that would have been of limited utility anyway. But we also wanted to make it so that those same, wherever we did cleanups to the APIs, and made it so that they were more future friendly and worked well on OS X, we also wanted to bring those cleanups back to OS 8.6 and 9.0 and make it so that developers didn't have to use the crusty, old APIs if they were developing a product for 8.6 or 9.0 and then use these new APIs if they were developing for X. It just becomes too many changes for developers to track, and if they're still targeting both pre-OS X and OS X, we wanted to make that as smooth and seamless as possible.

Hsu: Okay.

Ganatra: And so that's where CarbonLib came.

Hsu: Right. So, so then, so then did CarbonLib sort of translate between the new cleaned up APIs and the lower level older stuff?

Ganatra: Yes, yes.

Hsu: Okay.

Ganatra: Exactly, exactly. So CarbonLib, yes. So I mean, it had, you know, one of the biggest examples was getters and setters for, you know, for like Window Records or GrafPorts or things like that. And, I mean, it just had a pretty naïve implementation on the, you know, on the Carbon side-- or, I mean, on the CarbonLib side, the pre-OS X side. But on the OS X side, we could do whatever we wanted and it was hidden under this nice interface now.

Hsu: Right. So that it provided this API that worked across both platforms with different implementations underneath.

Ganatra: Exactly.

Hsu: Yeah, okay.

Ganatra: Yep.

Hsu: So then, so you had gone from that first sort of profiling task to then, how did you move from there to the CarbonLib development? Was there something in between?

Ganatra: There was, let's see, was there something in between? I don't recall-- I don't recall something in between, but I probably will when you remind me, but I don't know.

Hsu: Okay. <laughs>

Ganatra: I don't know. I mean, one of the first tasks that I took on after deciding that, okay, CarbonLib, we want this, we want CarbonLib on Mac OS 9 as well, was--

Hsu: Oh, you designed-- made that decision?

Ganatra: No, no, no.

Hsu: Oh.

Ganatra: I did not make this decision on my own.

Hsu: Oh, okay.

Ganatra: But it was sort of a--I was pitching that decision--

Hsu: Oh, okay.

Ganatra: And was happy that it was finally adopted. But there was a class library, it was called. There was a framework-- You know, before-- <laughs> before we had frameworks as, you know, given their modern definition on NeXTSTEP and maybe NeXTSTEP has had this definition forever, but before I became aware of the definition of frameworks, we--if you wanted to build a Mac application, you had to go and get a body of source code that did sort of the common things that a Mac application did, and you would just get the source code for it itself and build it in as part of your Mac application. And you would do some of the interesting bits to fill in what, you know, what was in the window and what the controls did when you clicked them and things like that, but the actual work of driving the interface and defining what it was like was all done by this class library. And so one of the popular ones at that time was this one called PowerPlant.

Hsu: Right.

Ganatra: Developed by Metrowerks. Metrowerks was a popular tool provider, a development tool provider for the Mac at that time and PowerPlant was a relatively popular class library that Metrowerks had developed as well. So I think one of my first tasks was actually building--was going through and building a few PowerPlant examples against this subset that would become, you know, the Carbon API.

Hsu: Oh. Right.

Ganatra: Just to kind of prove that okay, if I have-- if I want to do all of this work just on a, you know, on a pre-OS X Mac, what would that look like? Well, I would probably start with Metrowerks and I would probably have a PowerPlant app.

Hsu: Because that's what-- Yeah, because that's what most third party developers were using at that point was PowerPlant.

Ganatra: Exactly.

Hsu: Yeah.

Ganatra: Exactly. Most developers, third party developers, were using that. So, so where-- so within the PowerPlant project, instead of interfacing directly to the Mac's underlying interfaces.

Hsu: The Toolbox itself.

Ganatra: The Toolbox itself, I was going through the, what would later become CarbonLib. That would be my system interface for everything. And so if it wasn't implemented in CarbonLib, that meant that I had to figure out something else to do, either implement something in CarbonLib that supported it or, you know, for a lot of the symbols that were in the underlying system, CarbonLib just reexported them entirely. You know, if something was, already had a modern API and it worked pretty well or we decided that this API was good and we didn't have to touch it, CarbonLib itself would just take the underlying system's implementation and just have a way of connecting to it, and that's all, that's just how CarbonLib worked. But there were other places where if there was cleanup that was done to the API, then there was a little bit of extra code that was written to actually implement the cleaned up interface, but it was always in terms, for the most part it was in terms of what the old behavior was or the old way of doing things, if that makes sense, so.

Hsu: Right.

Ganatra: So, yeah, that was one of my first tasks was just take some popular PowerPlant apps and develop them against this thing that, you know, would become CarbonLib and see how far I could get to, you know, creating an application that used this new cleaned up API that still ran on Mac OS 8.6 at that time.

Hsu: So then at what point did you actually join the Carbon group in OS X proper from the classic Mac OS?

Ganatra: So I believe it was around when, around the time that 8.6 shipped. In fact, I'm not sure that I ever shipped-- I don't believe that I was responsible for CarbonLib when it first shipped on classic Mac OS.

Hsu: Oh.

Ganatra: And I'm pretty sure it shipped in 8.6, so I believe after Mac OS 8.5, I had transitioned over to reporting into Scott's [Forstall's] group, into Scott's org.

Hsu: Oh, okay.

Ganatra: Before 8.6 shipped.

Hsu: Oh, okay.

Ganatra: So, yeah, so what was that? '98? Something like that, I think. <laughs>

Hsu: <laughs>

Ganatra: So around then. Yeah, something like that.

Hsu: Okay. <laughs> So how-- was it very different culturally than what you were used to? Because I remember making that transition from OS 9 to OS X. And of course, I transitioned from classic Mac OS to Cocoa, so that was maybe a bigger transition in some ways.

Ganatra: Right, right.

Hsu: <laughs> But.

Ganatra: Yeah, it was. It was a big transition. It was interesting because that was sort of, that was when I became exposed more to the culture around NeXT as well, was when I had moved over from-- you know, before that, I was in this group that had existed before the NeXT acquisition, it existed for years. The org chain was all sort of old Apple. And when I moved over to Forstall's group, that was sort of part of the-- you know, that whole group reported-- also reported up to Avie, but it was a completely different management chain that reported up, and the management chain on that side was all NeXT-based. It was all people who had come over from the NeXT acquisition. And so, culturally, it was very different in that, you know, I wasn't used to, you know, a Vice President just walking into your office.

Hsu: <laughs>

Ganatra: You know, and asking how things were going, you know. <laughs> I think I had experienced it a couple of times with Steve Glass before, before that over the years, but it was never a, you know, Avie just is knocking on your door and has some questions about this API, you know?

Hsu: <laughs>

Ganatra: With Steve Glass, it was more of a, you know, "You're doing-- you're bringing up the hardware, you know, for this-- you're bringing up this hardware on our-- on System 7.6, how is that going?" And then two hours later, "How is it going? Does it work yet?" you know?

Hsu: <laughs>

Ganatra: How, two hours later, "Does it work?" "No, sorry. It doesn't work." You know, it was that kind of exposure and that was sort of what I was expecting. And that was terrifying all on its own, too, right, the VP drops in to low level engineer's office and, you know, you don't have good news for him. But anyway, but with Avie and with Bertrand, it was more of a, here's an engineer, who is in your office and now you're--but you just so happen to report to that engineer up the, up the food chain. But this is an engineer and you have to have-- you're having an engineering discussion now, right? You know, you're not going to wrap things up and give a non-engineering status to an exec; you're actually having an engineering conversation. And it was terrifying on the one hand, but it was fantastic on the other hand too, right? I mean, it was great because it was obvious very, very early on that Avie and Bertrand and Scott were just, were these super intelligent people and that they're there to have a conversation and we're all there to kind of figure out what is the best way to ship something and make something work well. And yeah, it might be--and Avie might have some strong opinions about how we need to follow more of a Unix development model or things like that, but it was all coming from hard fought experience and from his own lessons that he had learned in the industry as well. You know, it wasn't some MBA telling you what's what or somebody who read a book about programming and giving their opinion on it. It was great in that way.

Hsu: Yeah, like you knew he knew what he was talking about.

Ganatra: He knew what he was talking about. You couldn't bullshit him, you know? <laughs> And he was a great resource for bouncing ideas off of even though, even if sometimes the way it was, you know, the message was delivered was a little gruff, it was still, there was valuable insight there.

Hsu: Right.

Ganatra: So, yeah. It was terrifying and it was weird how the organization was so flat, as far as people coming into your office at any time or what have you. But it was-- it was different and it was good in a lot more ways, so, yeah.

Hsu: Yeah. Did you feel-- I mean, I know that there was from the classic Mac OS side, there was some resentment over the NeXT management chain and over, you know, sunsetting a lot of the older technology. What was your perspective on that and did you sympathize?

Ganatra: Well, not-- not really.

Hsu: <laughs>

Ganatra: No. I mean, I kind of felt like, on the classic Mac OS side, we had our chance to ship something great before then. You know, we had years and years and years to actually get our act together and ship something that didn't-- all of a sudden, these points were being thrown around that I had never heard before and that, in some ways amounted to religion, until there was actual data to support them. Right? Like all of a sudden, you know, a kernel that uses message passing for its primary means of communication is this horrible thing that will never be as efficient as, you know, direction function calls. And all of a sudden it's like, "Well, okay, I see your point. Yes, queueing messages and unqueueing and things like that are going to be slower. But why are we talking about this again? Like what are we--?" You know, it was that kind of thing. Or, or there would be questions about, you know, what toolchain we're using, and "Oh, my God, I haven't used GCC since I was in college and it's still garbage. And why does anybody use it. And, you know, and the code gen is horrible and blah, blah, blah." And it was always like, well we know from the Mac OS side that, you know, even the Adobe Photoshop engineers use Metrowerks. You know, they don't use the most shit hot compiler from Intel or from us to get the maximum awesome code gen. You know, they don't care. At the end of the day, they care more about having a toolchain that works really well and they will work around the deficiencies in code gen by hand rather than having something that's just this awful toolchain to work with but well, now you've got this great code gen. I mean, it was just sort of this-- It felt like a lot of the arguments that were being thrown around were religious in nature.

Hsu: Yeah.

Ganatra: And, you know, and they really weren't with-- they didn’t carry a lot of merit because I had never heard these religious objections before the NeXT acquisition. It's not like people were saying, "Oh, my God, I hope we don't acquire NeXT because they have the Mach microkernel and message passing is evil." Right. It was never anything like that. It was well after the acquisition was made and well after people had started looking into, you know, some of the underlying technologies and things like that that people had started making up their minds that, you know, this is awful and this'll never work and that, you know, you could never ship a desktop OS, you know, to consumers that has this level of code generation. Or, you know, these people all use Unix to install the stuff. I'm not going to tell my mom to use a command line. You know what I mean? It was like these-- it was these scare stories. It was this FUD, you know, that sort of didn't make a lot of sense to me and--

Hsu: FUD?

Ganatra: Fear, uncertainty and doubt.

Hsu: Oh, okay.

Ganatra: Right. It was a lot of-- it was a lot of this, you know, these concerns from, you know, this concern from very thoughtful people who never had an opinion about this stuff or never had strong opinions that I could tell, earlier, you know, who were happy to just kind of waltz along and watch Copland fall apart.

Hsu: Right. <laughs>

Ganatra: And now all of a sudden, now you-- now after we've made this acquisition of NeXT, now you've got this fire under your butt to, like, go and, you know, retrofit classic Mac OS with some sort of kernel underneath it? Like why didn't you do this two years ago?

Hsu: Right, yeah.

Ganatra: Like, where were you when we didn't have a solution, you know?

Hsu: Yeah. I worked with one of those guys. <laughs>

Ganatra: Yeah? Did you?

Hsu: <laughs> Yes.

Ganatra: Rene Vega?

Hsu: Yes.

Ganatra: Yeah.

Hsu: Yes.

Ganatra: Right. Exactly. And Rene, Rene's a brilliant guy.

Hsu: Yeah. He was very brilliant. <laughs>

Ganatra: You know. But, but at the same time, you could-- I mean, and I had many conversations with Rene as well, and they amounted to religion. Right? It was sort of the-- I mean, he was one of the big ones talking about how message passing is just inherently flawed for a high performance microkernel. And, you know, I'm sorry, but people's watches have a message passing microkernel on them now, so--

Hsu: <laughs>

Ganatra: Who was right and who was wrong? I didn't have a whole lot of patience for those discussions, and especially given the timing of those discussions and the effort that was going in, it all felt very suspicious to me, it all felt like religion, you know.

Hsu: Yeah. I, having been in the File System group, there was also, like, "Oh, why are we using extensions? Why aren't we supporting resource forks or these classic Mac--"

Ganatra: Yes.

Hsu: You know, or, "Why are we going to-- Why are we using paths?" I think that was one thing.

Ganatra: Right.

Hsu: It was, "Why are we using paths instead of, you know, the opaque Carbon Ref?"

Ganatra: Right. FSRefs.

Hsu: The reference—file--FSRefs, yeah, which are-- which are nicer because they don't, you don't get broken links and stuff.

Ganatra: Right.

Hsu: In some ways.

Ganatra: Right.

Hsu: <laughs>

Ganatra: In some ways, although you could create FSRefs that were, you know, that did have broken references as well, but.

Hsu: Yeah.

Ganatra: I mean-- <laughs> Yeah. No, I hear what you're saying.

Hsu: Yeah. The one that I never fully understood was the argument over the executable binary format. Like the Mach-O format versus the Code Fragment Manager PEF format.

Ganatra: Right, right.

Hsu: I never understood what the difference was there.

Ganatra: Well, I mean, I can say that the runtime, the PowerPC runtime was actually a lot worse. If you wanted to actually have-- if you wanted to create a system where you had multiple shared libraries that were, you know, sort of modular and were all used to kind of, to compose a unified system, the way the classic Mac OS worked, it was far worse for that, because there was an additional expense for anybody who wanted to make a function call into a separate shared library from the one that they were already in, there was an additional set of operations that you had to take to set up before you actually made that cross shared library call. So now, you know, and maybe that's just not a big deal, but what it really means is that if you're assembling an entire OS and if you have different organizations all building these pieces, you either take that hit on every single function call and then you're sitting there scratching your head and wondering how did we get into this position. Or, you do sort of the Mach-O style calls anyway where the environment is all set up for you anyway and you can jump locally just as quickly as you can jump across to other shared libraries as well. So, I mean, not only was the religion-- was there religion there and it was hard to tell, like, what the merits were, some of them actually weren't merits at all and it was just familiarity, you know. And people who I'd talked to about this specific issue of cross library calls, they, their answer was, "Oh, well. Just roll everything up into one system." And it's like, "Well, why is that a better solution than what's being proposed on the Mach-O side?" I mean, for a lot of these things, there were counter arguments, but you wouldn't hear them unless you were actually having the discussion with the right person to actually give those counter arguments. And luckily, I mean, to his credit, Avie and Bertrand didn't put up with too much of that. You know, they had enough of an understanding of how systems work and what was desirable in creating a system that they didn't put up with too much of that nonsense, even if it was coming from opinionated engineers, you know.

<laughter>

Hsu: Yeah. So then between-- So then, so now you're on the Carbon group.

Ganatra: Yes.

Hsu: And I guess this is like '98.

Ganatra: Mm-hmm.

Hsu: And then Mac OS X finally ships in 2001.

Ganatra: Yeah.

Hsu: So what, you know, what work did you do, you know, in that meantime to get the OS actually shipped from, on the Carbon side?

Ganatra: Yeah. So the big work I did was actually the File Manager.

Hsu: Ah. <laughs>

Ganatra: Was the Carbon File Manager. I took over ownership of that, I believe it was pretty much right away in '98 or '99, and started work on that and just making it so that we-- so that Mac developers had just a single API that they could use and they would work on UFS, you know, the Unix File System. They would work on NFS, the Network File System, this system developed by Sun. Or they would work on HFS+ and just making it so that these single APIs worked as efficiently as possible across all of those different file systems took quite a bit of work. And then also, in addition to that, just working with other clients higher up to make sure that they had, you know, that they understood what was going on and how the systems were different enough that, you know, calls that you used to be able to make that were close to free are now no longer free <laughs> so you better make some changes on your end to address that or you'll have performance problems.

Hsu: Right.

Ganatra: But it was, there was a lot of work in the Carbon File Manager primarily, but then also the Resource Manager and, you know, some work in threads and, you know, it was all-- it was all low level Carbon technologies.

Hsu: Right. Yeah. Now that reminds me that there were-- there was a low level Carbon group and then there was a higher level Carbon group that dealt with the UI. So, like, you were in the--what was that called, CarbonCore was that?

Ganatra: CarbonCore.

Hsu: Yeah.

Ganatra: Yep. I was in the low level Carbon group, CarbonCore. Reported to John Iarocci. Yep.

Hsu: And I remember you mentioned in the podcast that you did that you had worked with John-- John had been on the Copland team originally.

Ganatra: Right. Right. Yeah, John, I forget what he-- if he had managed the OS group in Copland or I don't remember specifically what his role was, but I know that he had come from Copland. I hadn't worked with him before that, but I knew that, you know, he worked on the Copland team and came straight over to Mac OS.

Hsu: Right.

Ganatra: Or to X, OS X.

Hsu: Yeah. And then there was the High Level-- was it High Level Toolbox or Human Interface Toolbox? I can never remember which one it is. <laughs>

Ganatra: I know. I think it's HLTB. I don't remember if it's HLTB or HIToolbox.

Hsu: Yeah. <laughs>

Ganatra: See, that's where I--

Hsu: <laughs> Yeah, I--

Ganatra: I always called it High Level Toolbox.

Hsu: High Level, yeah. I can't-- <laughs>

Ganatra: But yeah. But yeah, I've seen HIToolbox as well.

Hsu: Right.

Ganatra: So, yeah.

Hsu: So how much-- how much integration did those two groups have or?

Ganatra: Quite a bit. Yeah, we worked quite a bit together. We all sat sort of in the same general area and we, you know, Ed Voas was one of the engineering managers for that group and you know, Ed, John and I, we went out quite a bit, went out for lunches and things like that and-- yeah. Yeah, so it was a pretty close-knit team and we all worked pretty well together and I think we liked working with each other. I'm still friendly with Ed and-- <laughs> I haven't seen John in a while, but, yeah, it was-- it always worked pretty well.

Hsu: Yeah. And another thing I remember is that at that point, Carbon and Cocoa were both sharing this CoreFoundation layer and that was purely a C API, correct?

Ganatra: Right.

Hsu: And so there was work necessary to-- for Carbon-- to get Carbon to sit on top of CoreFoundation. Is that correct or--?

Ganatra: Yeah. Because, so now, this is where I don't remember the details. I believe that CarbonCore linked directly to CoreFoundation for collection classes and strings and things like that. But I do recall also that there were some parts of CarbonCore-- I believe that Carbon-- or, I'm sorry, that CoreFoundation actually had uplinks into CarbonCore itself as well for some things, and I don't remember-- I mean, it may have been, like, texting encodings.

Hsu: Oh.

Ganatra: Or things like that, I think that they had a soft link into CarbonCore from CF when doing some string translations to the Text Encoding Converter that lived in CarbonCore.

Hsu: Right.

Ganatra: And there may have been some other pieces too. But I believe that the layering was-- well, and this whole concept of layering, by the way, like this was a brand new concept too. Like, it had never-- I had never been exposed to it until the OS X, you know, sort of Rhapsody and OS X world, where it was a, okay, well the layering is really what reveals what your dependencies are. And, you know, so depending on how things are layered, that reveals, you know, that tells you, you know, where those dependencies are and how, you know, when you're building up a system from its core components all the way up, what do those dependencies look like? And they're not always pretty, but if you do a lot of work to manage that layering, it's going to pay off later when you have to-- in having a modular system where you can slip pieces in and out.

Hsu: Right.

Ganatra: So I know that the layering of CoreFoundation and CarbonCore was a little muddy because of that uplink.

Hsu: Okay.

Ganatra: But I believe that CarbonCore just linked to CoreFoundation.

Hsu: Okay.

Ganatra: Yeah.

Hsu: And classic Mac OS had never had any layers? You just, stuff was just--

Ganatra: And, yes.

Hsu: Just linking to other places willy-nilly? <laughs>

Ganatra: That's right. Exactly. That's right. That's why I bring it up is, I should have expanded on that. But yes, when you have a system that, where you understand the dependencies and where you have reasonably defined layers, it becomes much more possible to swap in and out new components or develop a new component in parallel, slot it in and have an understanding that it's going to work pretty well. But on a, you know, on traditional Mac OS, it was developed in a very organic way, let's call it, and--

Hsu: <laughs>

Ganatra: And yes, and there was just this jumble of dependencies. And so, you know, QuickDraw would call down into the Virtual Memory system for some things. And the VM had ways of disabling the cursor from moving during page faults. And you know, and so there were all these different, these interdependencies which, if you look at any one of them, they don't-- if you look at any one in a vacuum, it doesn't seem like the biggest deal in the world, but that's not how you develop these platforms. I mean, these platforms are really, you know, you're talking about dozens or hundreds of these types of exceptional dependencies. And so for each one of them, now you have to manage what these exceptions are and it just makes the job of having modular components much more difficult. I mean, you just don't have any modularity at that point. You just have this big jumbled mess and, you know, now you're trying to extract part of this jumbled mess and inserting another part, and still keeping the jumble as jumbled as it was, you know. So, <laughs> so yeah, I mean, and that was-- and this layering, initially when I had heard about it and we-- and there were discussions about it and there was this huge amount of work, I mean, that was part of the culture of coming over to the Rhapsody or the NeXT world. You know, there was all this work that went into making sure the layering was right and getting and keeping that layering and not violating the layering. And initially, it felt like, "Well, why?" you know. "Yeah, I mean, that seems like a good academic exercise." But it wasn't until, gosh, I wish I could recall. There had to be some-- There were some subsystems that we had replaced during OS X that went a lot more smoothly than I thought, where it sort of became obvious to me that, oh, well, this is why you care so much about layering. You know, you care about layering for years and years and years because some day it's going to save your butt. And it may not, for all those years leading up to it, but when it does, you're going to be very happy that you were, that you were managing it all the while. For me, one of the bigger-- the biggest examples was when we were doing early development of the iPhone and it was, "Okay, we really want to extract just these pieces from Mac OS X and we want to, you know, now build these other pieces brand new, but we want them to sit on these well-tested, well-functioning units and how do we do that?" Well, if we had a jumbled mess of a system, we would be trying to extract these pieces and managing all of these, you know, this long tail of dependencies, and it would have been much more-- you know, much more difficult to get to the point where we could just drop something on top of it without also having to, you know, manage all of this mess underneath. But, you know, when it came time to actually use these components and decide to use them and drop them in place, oh, my God, it came together, you know, very, very quickly and it was, you know, the credit goes to everybody who cared about managing dependencies before that. So, yeah, it turned out to be incredibly important.

<laughter>

Hsu: So then you're part of the Carbon team. OS X ships, 10.0 and then you have 10.1, 10.2. How long were you on the Carbon team?

Ganatra: I think I was on Carbon, I believe until 2002 when I started managing the-- the summer of 2002 was when I moved into management, engineering management.

Hsu: Okay. Right.

Ganatra: And so I started by managing the Mail team, the Mac OS X Mail application, email application team.

Hsu: Right. And so prior to that, you had-- so you had worked on, up until, up through the release of Jaguar 10.2?

Ganatra: Right, right. I believe Jaguar was the last version that I shipped.

Hsu: Right.

Ganatra: Yeah.

Hsu: And you were still working on the Carbon File Manager?

Ganatra: Yeah. I was working on File Manager. I think that there were-- were there other things too? I believe it was still predominantly File Manager. There was some work in Launch Services that I had been doing. You know, it was-- but it was still, like, low level Carbon, mostly File Manager work.

Hsu: Right.

Ganatra: Yeah.

Hsu: And then, so then you decided that you wanted to go-- you wanted a new challenge, you wanted to go into management?

Ganatra: Right. Right, exactly. I mean, I kind of, I wanted to move into management, I guess, to sort of expand-- expand, like, how much influence I could have.

Hsu: Oh, okay.

Ganatra: You know, over how things were developed. I mean, when you're an individual contributor, you can influence 100 percent of the thing that's right in front of you, or maybe 80 percent of the thing that's right in front of you, but your influence on to other aspects of the product kind of--it trails off pretty quickly. And so I wanted to have-- I wanted to know, well, what is it like to manage people and to have more of that influence on the engineering side too. Yeah.

Hsu: So then how did you end up on the Mail team?

Ganatra: I don't recall who brought it up. It may have been, like, one of our EPMs [Engineering Project Managers].

Hsu: <laughs>

Ganatra: Or it may have been Scott [Forstall]. I don't remember who had mentioned the idea that, you know, the Mail team might be looking for someone to take over management of it. Yeah, I don't remember specifically who it was, but I do recall talking to Scott and right away, you know, Scott saying, "Okay, go talk to the whole team and see how they feel about it, and, you know, and let's make the move if you're ready." So, yeah. Yeah, I don't remember specifically, though, who, you know, how that came about at the very, very beginning.

Hsu: And then at some point you also picked up Address Book. Or was that something that organically grew out of the Mail team?

Ganatra: I did pick up Address Book. I think Address Book I had picked up at the same time, I think it was like just three or four months later, so it was like early 2003 that I had started managing the Address Book team. And at the same time, Henri, Henri Lamiraux, who was previously managing Interface Builder and Address Book--

Hsu: Oh, right.

Ganatra: He had moved into like a second level management position, and so then I reported to-- where I was previously reporting directly to Scott managing the Mail team, I had now moved to reporting to Henri and managing Mail and Address Book.

Hsu: Okay.

Ganatra: So that was how that whole change happened.

Hsu: Right.

Ganatra: It was early 2003.

Hsu: Right. And so there were several engineers on Address Book that later joined you on the iPhone?

Ganatra: There were, yes. Yes, there were a couple of engineers. It was not a very big team, the Address Book [team].

<laughter>

Ganatra: I think it was two engineers and one tester.

Hsu: Okay.

Ganatra: When I started managing the team. I think it was Scott and Vince.

Hsu: Scott Herz and Vince DeMarco.

Ganatra: And Vince DeMarco.

Hsu: Yes. Who had both been on Interface Builder before, I recall.

Ganatra: Who were both on-- Right, on--

Hsu: Working for Henri directly.

Ganatra: Working for Henri, right. Right, exactly.

Hsu: <laughs>

Ganatra: And so, yeah, so I picked up the, you know, those two, they were the engineers on Address Book and I had the Mail team as well. And by then, I think the Mail team-- Well, it was pretty early in the

Mail team development, and so it was still, I think it was like six or seven people, plus three on Address Book. There were two engineers and one QA, Brett Neely, as well.

Hsu: Oh, yes, Brett.

Ganatra: Yeah.

<laughter>

Hsu: So then, oh, man. Wow, I've blown through--

Ganatra: Are we out of time?

Hsu: Well, no. We still have 40 minutes, right?

Ganatra: Okay.

Hsu: How much time do we have?

[Crew talks]

Hsu: Roughly? Okay. Just let me know when the time comes. But, well, I will try to see how much I can get through. <laughs>

Ganatra: Sure.

Hsu: But, okay. So, so you're managing these two teams. You know, you're managing Scott and Vince directly. You're reporting to Henri. At some point, it's like that whole sort of organization just becomes part of iPhone. Like, how did that happen?

Ganatra: So it wasn't the whole-- Oh, you're saying Scott and Henri and me and Scott Herz, like that, how did that become?

Hsu: Yeah, like the-- Yeah, well, it's like-- Like, yeah, exactly. Like Scott Forstall, Henri, you, Scott Herz and Vince, too, I think, right?

Ganatra: Vince, yeah. Vince had come later, yeah.

Hsu: Oh, but it was later. Okay.

Ganatra: Yeah.

Hsu: So, well, maybe we should just go back to, like, how did you get into the iPhone project?

Ganatra: Okay. So one day-- I mean, this was something that we had had discussions about, just sort of hallway conversations, and you were probably in on some of those too, where everybody just pulls out their latest phone and we're all kind of looking at them. And, well, this one sucks for that reason and that one is terrible for this other reason. And, you know, just kind of going through. And, "Well, here's what's cool about," you know, "about this phone." At that time, it was let's make them as small as possible. You know, who has the smallest phone.

Hsu: Right.

Ganatra: That was the competition.

Hsu: <laughs>

Ganatra: But there was always-- but there was already water cooler talk about, you know, "Well, we made the iPod." You know, I mean, iPod was so much better than any of the MP3 players that came out before-- that came before it, you know, and we know all of these phones are terrible. What if we made a-- like, why don't we make a phone? But it was never anything more than, you know, it's not like Steve was sitting there and-- Steve Jobs was sitting there and said, "Well, that's a great idea. Let's go do it." You know, it was just, you know, water cooler talk, right.

Hsu: Mm-hmm.

Ganatra: But, but I'm sure that Scott Forstall was in on some of that too, with just hallway conversations and chatting about what we were doing.

Hsu: How early was this? Like 2004, 2003?

Ganatra: Yeah. I mean, those conversations had been happening, 2003, 2004.

Hsu: Yeah.

Ganatra: That kind of thing.

Hsu: And wasn't there, like, you know, this, the Motorola ROKR, that Apple had partnered with Motorola to do this iTunes phone?

Ganatra: Yeah. That came in the fall of 2005.

Hsu: Okay.

Ganatra: And so, so what happened was early in 2005, it was shortly-- it was right around the time that Tiger had shipped. Mac OS X Tiger, which was, I think, 10.4.

Hsu: Yeah.

Ganatra: Scott came into my office and closed the door and said there's this phone project that's starting up. Would you, how would you feel about managing the software side or managing the software team. And so, I was like, "Well, yes, that sounds great, but I have a--" You know, by then, the problem, the sad part was that the Mail team was, I had sort of built up the Mail team. You know, we shipped Tiger which had really, really good reviews. Like the email client in Tiger, we had great reviews for it. We had already-- I felt like we were kind of firing on all cylinders because before Tiger was even done, we had already gone through this exercise of coming up with some demos and we started demoing features to Scott to show him and say, like, "Here's what we want to do for the next release." We don't even-- We're not even completely done with Tiger yet, but we already want to start figuring out what we're going to do for the next release, and so we had these demo meetings and they all went really well. And then all of a sudden, Scott comes into my office and says, "How would you like to manage the [phone] team?" We talked about it and well, what does that mean for, you know, do I keep managing the Mail team or is this a full-time thing? It became clear it was a full-time thing that I, you know, need to-- I can pull some people off of the engineering teams and, you know, to go and start on this. And, you know, that in the HI hallway they already have designs for it, so I would very likely have to move up there. Just kind of working through some of those details and--

Hsu: Okay. So work had already been going on up there in the HI, in the HI team?

Ganatra: Yes. Right, right. Work was already going there. I hadn't seen anything. All I had heard from Scott was just work had been going on and--

Hsu: Right.

Ganatra: I knew that they were hiding things up there because they're always hiding things.

Hsu: Right.

Ganatra: And, you know, who knows what they're hiding, at any point. So they do a good job of hiding what they're working on, I guess.

Hsu: <laughs>

Ganatra: So, yeah, so Scott mentioned that to me. I think that was April of 2005.

Hsu: Wow.

Ganatra: I made the decision to go for it and then took three engineers-- two engineers from the Mail team, one engineer from the Address Book team. By then, Vince had already gone. I think it was Alex Aybes was-- It was Alex Aybes and Scott Herz were the Address Book engineers.

Hsu: Okay.

Ganatra: So, you know, [I] left the Address Book team with one half of their engineering staff.

Hsu: <laughs>

Ganatra: And pulled two people from the Mail team, and then the four of us, me and three engineers, went up and moved into the HI hallway. And that was when I first started-- that was when I first saw designs and you know, and started working with the team and working with HI to kind of fill things out.

Hsu: Right. And at that point, what were-- what did the designs look like?

Ganatra: They, so the very first designs, they looked an awful lot like what you see today-- like when you look at the sort of the iconic iPhone screen with Springboard and, you know, the grid of icons, that was by and large fleshed out by then.

Hsu: Okay.

Ganatra: You know, there was the status bar along the top. Grid of icons. I believe there was even the dock-looking thing on the-- for the bottom icons. You know, and there was a phone app and everything. And, you know, and so on this-- as part of this design, it wasn't just still pictures. What the HI team had done was they did a lot of work to drive these Macromedia Director demos. And so that the-- Director was this animation, this easy animation creation app that you can use to kind of-- that the HI team had used to sort of create these animations which were transitions between the apps or an animation to bring up the keyboard or animations to show scrolling within the address book lists. Different things like that. So what they had was this pretty well fleshed out demo of--so it was a Macintosh. And sitting attached to it was the screen that had a display on it and it had touch input on it. It was about the size of a tablet, though, and so pushed down into a corner of it was Springboard-- was the Springboard UI. And so you could actually interact with it down in that bottom corner, tap on it. You know, something that looked like Mail would come zoom up and show you what it looked like to be viewing a mail program. You know, you hit this other button that was sort of the equivalent of the home button and tap on something else and then another program would come up. So, so it felt-- it felt quite a bit like the phone experience ultimately would, but it was all just done in Macromedia Director, all running on a Macintosh. It was just a script running on a Macintosh. And the designs were there. So, yeah.

Hsu: Did you ever get to see the design that had been based on the iPod, like, scroll wheel? Or was that long before?

Ganatra: The scroll wheel, that was actually, that came later.

Hsu: Oh, it came later.

Ganatra: Believe it or not. That came later. And now they may have been working on designs before that too. But the designs that I saw were for the P1 project

Hsu: Oh.

Ganatra: Which was-- which was something that came later, after we had started on Purple.

Hsu: Okay, so then, yeah, so then there were two competing projects going on?

Ganatra: Eventually, there were-- So for a lot of 2005, we were working with the Human Interface team and just sort of really trying to flesh out, like, well, what does this mean to, you know, to zoom on a photo. Or, you know, what-- when you say Safari, is it all of Safari? Or when we say Mail and let's say I have 10,000 messages in an inbox, am I going to really be able to scroll across 10,000 cells in order to see that, the very first message, or are we going to cut them off? You know what I mean? There's just, there's a near infinite number of questions like that to kind of flesh out, like what are the boundaries of this user experience and what does it look like and how does it operate at those boundaries so that when we're going in and implementing things we have a better understanding of what we're doing. In addition to that, there was a lot of work to actually just flesh out, to actually prove to ourselves that on ARM class hardware, can we actually get 60 frames per second of this-- of the Address Book animation scrolling? You know, because we understood that there were multiple layers there and multiple pieces were all moving simultaneously, but can we actually do that with a 200 megahertz processor or--and the GPU that's attached to something like that? Or are we just going to fall way short? You know, how close can we get? So there were different things that were-- I mean, there was a whole lot of work that was happening on the software side for many months, you know, including decisions around what underlying frameworks we use, what language are we going to use, you know, what runtimes will we support, things like that. But anyway, after that work had been going for several months, there was, we had this meeting, there was this meeting that Steve Jobs had called where Tony Fadell was there, Scott Forstall, John Rubinstein, sort of our management team was all there, and in that meeting, that meeting basically was the kickoff meeting for what would become P1. What Steve had said was that he sees a lot of great progress happening on Purple, on the Purple project, which is what the iPhone, what would later become the iPhone was known as, but he's worried that we're going to take too long to ship it, so we need some sort of interim product to get out there to, to help us drive sales on something in the meantime and maybe we can actually learn about phone development along the way too. And then in a very orchestrated way, Tony Fadell just pulled out-- happened to pull out these schematics of a click wheel phone, you know, that had a bigger screen on it and that was sort of how-- that was where the P1 and P2 monikers came [from].

Hsu: Okay.

Ganatra: But before that, there was just Purple and there was nothing else.

Hsu: Oh, okay.

Ganatra: But P, you know, it became P1-- P1 became the click wheel iPod-based phone and then P2 was the thing that we were then working on.

Hsu: Right. Which was just the continuation of Purple.

Ganatra: Right. Exactly.

Hsu: Okay.

Ganatra: It was just a rename of the-- of Purple.

Hsu: Oh, okay. But then, but that predated the P1 to some-- to a large extent?

Ganatra: Yeah, I think so. I'm, it's--

Hsu: Although maybe there had been initial prototypes with the click wheel--

Ganatra: Right, that I just--

Hsu: Done by HI before, right, HI before.

Ganatra: Right. Right. And I'm not even sure, you know, maybe in the iPod org, in Tony's org, maybe they had pitched a phone a year or two before and I just, I didn't know about it because I wasn't, you know, there. It was--

Hsu: Right.

Ganatra: Yeah. That was-- So there may have been things that predated it, but I'm not aware of what they were.

Hsu: Right. And you had mentioned there had been discussions over what language to use for Purple, what frameworks to use. I think was it-- Well, I think you mentioned that there were three options for what, you know, what the decisions were.

Ganatra: Right, right. There were the three-- eventually, we had sort of narrowed it down to three, the three best options. And this is for the runtime and then for what frameworks we would use for largely for implementing the user [interface], the look and feel of the iPhone. The first option was taking existing Mac OS-- an existing Mac OS framework that's used for drawing widgets and controls and things today, called the AppKit, and porting that over to make that the new, the basis for all, you know, widgets on the phone itself. That was one choice. A second choice was web technologies. So using some of the things that were being developed by the Mozilla group for Firefox were, there were these cross platform web-based languages and environments that were starting to be created and were starting to get popular, but-- And the big draw with all of those was that you could use web technologies that a large number of engineers understand and you could implement things on across multiple platforms.

Hsu: Right.

Ganatra: So those were starting to gain popularity and obviously the web team was pushing very hard for using those technologies.

Hsu: Right. And OS X also had this Dashboard thing that also used the web technologies that had these little app-like widgets.

Ganatra: Yes, yes. Yes. Thank you, thank you, yes. So OS X, I believe in 10.4, or earlier in 2005, had shipped with Dashboard, which were these web--if you look at them, they look like web apps in a way. Each one is just its own fully formed UI. You know, each one is sort of responsible for a specific task. They tend to look very nice. They have very good production quality as far as the look and feel of them. More the look than the feel, but.

Hsu: <laughs>

Ganatra: But that was technology that was newly developed at Apple and that was developed by the web team as well, and so they were, the web team was pushing very hard that, well, we should just implement all of our apps and the whole look and feel using these web technologies, because look at what we can do with Dashboard. Let's just expand it and make the whole system that way.

Hsu: Right. And so everything would be like JavaScript and HTML and CSS.

Ganatra: Right, exactly, exactly. And then the third choice was stick with Objective-C, you know, stick with sort of the native runtime that we had known from Mac OS but develop our own brand new framework, right.

Hsu: That the AppKit, well, had already-- the AppKit had used all the way back to NeXTSTEP had used Objective-C.

Ganatra: Right, right. Objective-C, right, exactly, that was sort of a traditional NeXTSTEP, OpenStep technology, was the Objective-C language. So stick with that because it was familiar. Really, the-- but create something new that was tailor-made for a phone or for, specifically for a touch device, not a device where you have a mouse, where you can control it sort of remotely, but really touch is front and center and we want to make sure that everything related to touch works really well in this framework.

Hsu: Right. And that-- And by that point, the decision had already been made that this would be a multi-touch based device?

Ganatra: Yes. Yeah, absolutely.

Hsu: Right.

Ganatra: From the very beginning, it was decided that that was going to be the primary interface, was the display itself would be touch capable. And so eventually we had decided after a number of meetings, we had decided to take the third approach, which was create a brand new framework from scratch, but really what we wanted to do at the same time was base the patterns that were used in this framework on AppKit. Because there isn't just the language and there isn't-- I mean, there's the language is one thing and there's conventions around how to create the interfaces is another thing. But we really just wanted to make sure that-- forgive me, I'm not saying this really well-- but we really wanted to make sure that developers who came in and started implementing phone apps could feel as comfortable as possible right away. And we knew that internally, at least, we had a lot of engineers who understood AppKit and Foundation and understood delegation and some of the rules around memory, you know, memory management and things like that, and we wanted to keep as much of that as familiar as possible so that there wasn't just this huge learning curve for anybody coming in where they wouldn't know how to do anything. We really wanted to make it so that they had an understanding of the foundational components that had already been there on OS X. We wanted to make sure that they leveraged that knowledge and only had to learn the new, the new-new parts that were unique to development of a phone app.

Hsu: Right.

Ganatra: And so that was how UIKit was created. It was largely, if you look at the interfaces, they look like they could sit side by side with AppKit interfaces and they would feel very-- if you've been doing development work in AppKit, switching over to UIKit felt like a very natural change to make, and that was very deliberate.

Hsu: Right. And so in other words, bring the Cocoa programming paradigm, just migrate that more or less over to the iPhone essentially.

Ganatra: Exactly. Exactly.

Hsu: Yeah.

Ganatra: Keep-- the Cocoa programming paradigms, keep those. Change some of the details around, the names of the frameworks themselves and some of the classes. But just, but keep the spirit as much as possible.

Hsu: Right. And there was never any, at any point in time, any discussion of using Carbon rather Cocoa as the basis?

Ganatra: No.

<laughter>

Ganatra: Interestingly, no. No, not at all. But by then, I think, I think it was pretty clear by then, you know, what Apple was investing heavily in. And Carbon was, I mean, Carbon was critical to actually making the migration over to a new OS. But even by 2005, it was pretty clear that if you wanted to have an app, a successful app that had really great production and worked really well on Mac OS X, you had to write a Cocoa app. You know, you could write a Carbon app, but it's just going to-- it's going to feel a little-- it's going to feel different enough and strange enough that you're not going to be able to have a really great user experience around it and have an app that works as well as the rest of the system does. If you were developing a Carbon app, you would have an app that has, you know, a subset of the functionality of other apps on the system. And so Carbon had sort of done its job by then, in my opinion anyway, and sort of helped us cross that bridge that we needed to cross. But it was really, you know, Cocoa and the Cocoa APIs and just the Cocoa paradigms were really, had taken hold and there was no going back or changing from that.

Hsu: Right. Right. But then you mentioned, there was the first option was to actually bring AppKit itself, which was the key component of Cocoa and make that the phone. Why was that decided not-- I think you've told the story that the manager of the Cocoa or the AppKit team, Ali Ozer, who was my manager, incidentally, actually made an argument that you should not use the AppKit for the phone.

Ganatra: Yes. Right, right.

Hsu: Why was that? <laughs>

Ganatra: Yeah, well and that's, and it's funny, I mean, that as engineers, you a lot of times, you know, engineers are very sort of proud of the work they've done and they want to-- we want to-- believe that the things that we've created are-- can be suited or can be used for anything, including things that we never anticipated they be used for. And a lot of times that's just not true, <laughs> you know. But I think, some amount of the pride of ownership and pride of development kind of clouds your judgment in some ways and makes it so that you, you might think that the thing that you've developed is far more capable than it actually is. But that's the-- but to Ali's credit, that's not Ali. <laughs> And that's not Kristin. I mean, I think that they had a-- they have a very good ability of sort of separating what their opinions are and any sentimental attachment they have from the actual task at hand and so because of that, you know, if anybody was going to make the argument that we should use AppKit for future phone development, it would have been the manager of the AppKit team, you know, if they operated like your average engineer does. But to their credit they don't, and so <laughs> they were well aware of all the ways that the event system and that menus and that mice and that, you know, all these things that were very essential to how desktop computers work, they were so embedded into how AppKit handled events and did so much of what it does that they just, they themselves, Kristin and Ali, were making the argument that it's just not well suited for something like touch or for multiple layers on the screen or things like that. So they, you know, the fact that they were arguing against using AppKit to me was the strongest argument that okay, I mean, these are, they're super smart people and they are-- and they understand their technology better than anyone. And so why would we <laughs> try to use AppKit when the experts are saying we really shouldn't.

Hsu: Right. And just to clarify, that's Kristin Forster who is also on the AppKit team.

Ganatra: Right.

Hsu: Who was also in that meeting.

Ganatra: Kristin, yes.

Hsu: Yeah.

Ganatra: Exactly. Kristin Forster, yes, in the AppKit, yeah, was in that meeting. And I believe she did the majority of the research as well.

Hsu: Oh, okay.

Ganatra: So, yeah.

Hsu: So I guess the decision to use Objective-C was once you had decided not to go the web route, Objective-C was always the alternative because it was either use AppKit or to write a new UI framework using the same Cocoa technologies, which included Objective-C.

Ganatra: Yeah. You know, I guess we hadn't really, we didn't really strongly consider other, you know, like pure C++ or other languages. I mean, maybe it's just that we're, everybody who was in the decision making were fans of Objective-C--

Hsu: <laughs>

Ganatra: And so we didn't see a reason to change it. But I like Objective-C a lot and I think that it's, the fact that it worked on 386-class hardware back in the late eighties, on NeXT hardware, just made me feel like, well, we're not going to-- it's not going to be a performance issue that, you know, with the language or the runtime that makes our jobs difficult if we choose Objective-C.

Hsu: Right, right.

Ganatra: And we had enough engineering support within Apple and it was just sort of a natural fit that this is what we would use.

Hsu: Right. So not a lot of discussion about that. Yeah.

Ganatra: Yeah. Not that I recall, no.

Hsu: That's just what Cocoa programmers use, so if we're going to make the phone based on Cocoa, we're going to use Objective-C.

Ganatra: Yeah. I think so. <laughs> Yeah.

Hsu: Earlier you also mentioned, you know, the developer story. But you weren't thinking of third party developers, right? Like, there was no idea that this would be opened up to third party app developers at that point in time?

Ganatra: Correct. Correct, correct. At that point, it was really, well, when I say developers, at the same time that we were doing these investigations and some of these proof of concepts and things like that, I was also staffing up the team. And so, it was, it was very plain in my mind that okay, we're going to bring these-- I need to bring these people in, new people in, and I need them as productive as possible as soon as possible. And so what better way to do that than to give them tools that they're already familiar with and a language that they're familiar with. So when I say developers, really what for me, that was the big priority, just to be able to bring people up online, you know, get them up and working and functioning being productive as quick as possible. And this was internal Apple engineers and new hires that we may bring in too.

Hsu: Right.

Ganatra: But yes, there was no consideration by me, anyway, for third parties at that point.

Hsu: Right. And originally, some of the, you know, what look like apps were just web, done in the web technologies and WebKit?

Ganatra: Yes. Yes, and so--

Hsu: And there was a whole other group that was doing that?

Ganatra: Correct, correct. So that was, my peer-manager was a guy named-- is. He's still named that.

Hsu: <laughs>

Ganatra: Still named Richard Williamson. And he was managing-- he came from the Safari team, the Safari and WebKit team. And on their end, they'd had some widgets that were going to ship as part of the phone itself, including Weather, Stocks, Traffic, I believe it was called or Maps back at the time.

Hsu: Oh, Traffic, uh huh.

Ganatra: Something like that. And maybe one more, and I don't remember what it is. But anyway, these were things that were already largely implemented as widgets on Mac OS X.

Hsu: Oh, the Dashboard widgets?

Ganatra: Yeah.

Hsu: Yeah, because they look exactly like the Dashboard widgets.

Ganatra: Yes, exactly. Exactly. And at that time, they just brought them over and it was just, look, you know, you-- the design had the Dashboard widget. The implementation is the Dashboard widget.

Hsu: <laughs>

Ganatra: You know, boom, right there. It's done.

Hsu: <laughs>

Ganatra: You know, that kind of thing. And so, so, yeah. So there was a, at least for some of those Dashboard widgets, that was sort of the early proof of concept around well, what if we did use web technologies for some of these things? You know, I think at least very early on after we had made the decision to not go with web technologies, the door still wasn't completely closed on whether we were going to use Web or not, and Dash-- and those Dashboard apps were sort of the last, you know, the last attempts at making it so that we use web technologies for these apps instead. Eventually we never actually shipped iPhone 1.0 with those implemented as Dashboard widgets, though, and the reason is just performance. I mean, performance and modifying those, the assets, just became too big of a problem. And so, when you're at a point where you could launch the Mail app and bring up Mail and start responding to your Mail quicker than you can launch the Weather app to look at what the weather is in Cupertino, you kind of have a problem. And to management's credit, they understood that, a few months before we shipped and decided that okay, we need to just turn these into native apps instead.

Hsu: Only a few months, that's a quick turnaround.

Ganatra: It's pretty [quick], yeah.

Hsu: <laughs>

Ganatra: Everything was pretty quick turnaround in those days, so, yeah.

Hsu: <laughs>

Ganatra: To Richard's team's credit, they did knock those out pretty quickly.

Hsu: Oh. So they reimplemented those apps natively.

Ganatra: Yes. Right, right.

Hsu: Not your team.

Ganatra: Not my team.

Hsu: Oh, okay.

Ganatra: Yeah. They reimplemented those apps.

Hsu: Wow.

Ganatra: As native apps themselves. Yep.

Hsu: So which apps were you in charge of?

Ganatra: I was in charge of--

Hsu: Well, you were you in charge of more than just apps, right? You were in charge of the entire user experience.

Ganatra: Well, and there was Springboard. Springboard which is sort of the puppet master for all of the apps on the system, and the very earliest version of UIKit before it moved over to become its own thing and become something that third parties used, my team developed the early versions of UIKit as well. It was just an <inaudible>.

Hsu: When that was just internal use?

Ganatra: Exactly. Exactly, when it was internal use.

Ganatra: As far as the apps go, Camera, Photos, Mail, Contacts, Calendars, Settings. Later the App Store and the iTunes Music Store. Videos, the music player. We implemented a lot of the-- you know, the clock app. I mean, like, a lot of-- a lot of the apps that you saw on that first screen, probably like 70 percent of them were, my team developed those.

Hsu: Wow.

Ganatra: Yep.

Hsu: <laughs>

Ganatra: Yeah.

Hsu: That's a lot. <laughs>

Ganatra: It was-- it was a lot. It was a lot.

<laughter>

Hsu: I think you told a story about how when you first signed on, your team did not have access to the UI designs and only you did.

Ganatra: Correct.

Hsu: And you had to go and somehow tell them what you had seen without-- <laughs>

Ganatra: Right, I--

Hsu: Without really telling them, you know, you couldn't give them the actual design, you had to, like, redraw it from memory or something?

Ganatra: Correct.

Hsu: <laughs>

Ganatra: For a few days, I had disclosure to see the UI as the manager, but the engineers on my team had not been disclosed yet about--.

Hsu: And this was like right at the beginning when they had just joined the team?

Ganatra: This is right-- Yes. Right, right at the very beginning. And so it was probably for the first three days or four days that we were in this arrangement where Steve Jobs had to personally approve not only everybody who was on the team but then also everybody who had access to the user interface. And so, and I remember I had discussions with Scott Forstall about this too, where I had access to see the designs but my engineers who are actually doing the work here <laughs> didn't have access. So what should I do? You know, like, should they just go on vacation for three days? That doesn't seem right.

Hsu: <laughs>

Ganatra: You know, we're busy. We need to get busy with this stuff. And so, I don't remember if Scott came up with it or if I came up with it, but eventually, <laughs> the solution we had was, so we had two of my engineers were in one office and in an office right next to them was a computer that had the designs on it. And so, any time they had a question about the designs or some aspect of the design itself or something, what I would do is go into the-- and they weren't allowed in the room with the computer that had the designs on it. So I would go into the room with the computer that had the designs on it, play with it for a little bit, try to commit to memory everything that I saw, and then I would come back over and on the whiteboard I would draw, and I'm a terrible whiteboard drawer anyway. And then I would draw an approximation of how things worked and how it behaved and how things should be. And so, we did this for three days. And the whole time we were doing it, we were all perfectly aware of how absurd it was.

Hsu: <laughs>

Ganatra: But, you know, it was-- that was what we had to do and we had to show demos, either that Friday or the following Monday. And so, we had work to do and we had a deadline to meet, and this is the way to meet the deadline was--

Hsu: Oh, man.

Ganatra: So, yeah, it was a little silly for a while there.

Hsu: <laughs> It's like you're a corporate spy within your own company.

Ganatra: Right, right.

<laughter>

Hsu: That's what it sounds like.

Ganatra: Exactly.

<laughter>

Ganatra: Yeah. Yeah. Somehow and I'm, yeah, and I'm reverse engineering what that design is and, yeah, it was very silly. Very silly.

Hsu: Were there other instances of, just being hampered by the siloization, by not being able to see the hardware, for instance, or, you know?

Ganatra: I'm sure there were. And I mean, I think that we were hampered in so many ways. I just can't remember specifically how things hampered us, now. I mean, I feel like whatever we needed, whatever information we did need, you know, about the hardware or certain aspects, we managed to get the information that we needed. It was sort of, everything was need to know, of course. And the things that we did need to know we did eventually find out. So it wasn't too bad for my team. I mean, I think it was probably a lot worse for other groups, just because we had access to the internal UI from the very beginning. And so it was easier for us to go to the hardware group and say, "Hey, you know, what are we talking about as far as RAM here? You know, are we still talking 8 megabytes or are we talking 128 megabytes?" Like, you know, we could have those discussions. It was easier for us to have those discussions with other teams and not be hampered by the lack of information than it was for those teams to work with us.

Hsu: Oh.

Ganatra: So, you know what I mean? Like, I feel we were--

Hsu: So you were more privileged access than other groups.

Ganatra: I feel like we had more privileged access and so we were a little bit luckier, even though it meant-- but what it did mean was for everybody who came into our team, we had to let them know how

privileged they were and how little other groups knew about all of this information or knew about the designs and what we were considering and things like that. So, so it was hard when we would talk to other groups, and I know that other groups were impacted greatly, but we were less impacted, fortunately.

Hsu: Right.

Ganatra: Yeah.

Hsu: I mean, is that just a-- Do you think that there's a difficulty there or that does-- do you think that in some ways that negatively impacts the way products are developed when you have this much secrecy between the individual groups?

Ganatra: Oh, yeah. Absolutely. I mean, I think it negatively-- The problem is that it negatively impacts management in a lot of ways because, I mean, what I'm saying is the burden really falls on management to try to do everything possible to minimize where that causes slowdowns.

Hsu: Because it's the managers that have the access, but their engineers don't?

Ganatra: Well, I mean, a lot of times even managers didn't have access too.

Hsu: Oh.

Ganatra: You know, I mean, it was sort of, up the chain, people didn't have access. But really, I guess I'm just saying, like, I mean, any type of development work that you do comes with constraints, right? And it's sort of management's job-- one of management's jobs to the employee you're clearing bull-- you know, you're shielding your employees from bullshit and you're also trying to make sure they're as productive as possible. So when you have a new constraint slapped on a group like this, and it was a huge deal, but I mean, and it's really up to management to find the best ways to work around that or to communicate up the chain why a team needs more access than they have. But you know, the thing is that a lot of times you can get an awful lot done in a vacuum as long as you understand a couple of different constraints. You know, you may want access, you may want to see it just to see something. But really, you don't need access in a lot of ways. And just like we didn't need access to the hardware, as long as we understood, okay, this is-- these are the screen dimensions. This is the processor. We can approximate its behavior in other ways using these certain techniques, ultimately we did not need to see the hardware while we were doing development in parallel. It would have been awfully nice, but we didn't need to. And so I think it's the same with the software, with the designs too. You need to find the information that you need to know, but ultimately, a lot of times people just want the information just to have it and they may not know why they need it. And so, some of it is just a matter of, "Well, we need to see the designs in order to be able to

create something." And it's like, "Well, why? What do you-- What specifically are you running into that you need?" And once you kind of, once you ask a couple of those questions that way, you can really find out does somebody need access or not. So I don't think-- I mean, yes, obviously it did slow things down and it was a big pain to a lot of other groups, more than to our team. But at the same time, we were able to ship the phone and get the big splash that it got and, the first day that, after demos, after Steve Jobs demoed it, it was on the front page of newspapers. And, I had never been part of a project that had that before, and a lot of that was because of the secrecy around it as well.

Hsu: Right. Yeah. Was there, you know, was there any tension between, Scott's organization and Tony Fadell's organization, especially because at one point you were working on these two competing projects, like P1 and P2?

Ganatra: Yeah. Yeah, there was. There definitely was. I mean, I think that the iPod team, I mean, if I can characterize sort of the attitude that the iPod team had, was that, I mean, they had shipped the last successful embedded product, so in some ways, it was sort of their right to ship the next embedded product that Apple was going to do. And so, who, and I'm sort of playing the part of a, of an iPod engineer, well, "who the hell are these computer people to come along, these desktop people, who've only ever shipped--" in the minds of some of-- And I can say this because a fair number of them are my friends and we're still friendly to this day, but "all these desktop people have done is ship piggy, you know, porky desktop software that barely works on a, on a G5, on supercomputer class hardware. Now we're going to trust these people to implement something on the phone? Like, why do we even think that's a reasonable plan to begin with?" You know? I mean, so I can understand where they were coming from and I had people in meetings say, "You do not understand the first thing about embedded development." <laughs> And, and sadly, they were kind of right, but at the same time, I mean, I think that we, we let that fear of the unknown just make it so that we were extra careful and that we really understood what the requirements were for developing these-- developing a phone and what does it mean to get technical approval from a phone carrier, from a network carrier and things like that. So, yes, we didn't know what it was like to develop a phone from the beginning. But at the same time, we were developing a phone that was unlike any phone that had been developed before. It was closer to a computer in the end. And so, really, it was the right decision in my mind. I may be a little biased. But it was the right decision to have people with a computing, a desktop computing background be the ones who work on this and scale down a desktop computing experience rather than taking an embedded experience and trying to expand, you know, expand it out. You just end up with, with a software stack, and I think that we've proven it-- we had proven it in later years. You end up with a software stack that is more tailored to other embedded devices and other embedded applications if you come to it from a desktop perspective as opposed to, you know, in the embedded world, things are so one shot and one off and you know, get this product done and we'll worry about later products later. Whereas what we did was a lot more akin to computing, where we have one software stack and it can test the capabilities and it can run on multiple pieces of hardware based on what those capabilities are.

Hsu: So I'm trying to remember. So at what point did P1 sort of go away and P2 became the phone that you guys were going to ship?

Ganatra: So I believe that that all happened-- So I think P1 kicked off-- I could be wrong on these dates-- I think P1 kicked off in late 2005, like the fall of 2005.

Hsu: When it started?

Ganatra: When it started. And I think by the spring of 2006, it was sort of gone.

Hsu: Oh, wow.

Ganatra: It was, we hadn't been talking about it as much anymore.

Hsu: So it was like six months or something.

Ganatra: I think it was. I don't think it was any longer than six months. Yeah. I think that it was about that long. I do, I know that, once P1 had sort of become plan of record, that was sort of, that became a rallying point for our team, you know, on the P2 side, to actually try to deliver features and functionality before it was available on P1 to kind of, to--there was this perception that Purple and, that what was later named the P2 project was kind of the science project and it's going to take some time to actually, have the rubber hit the road and actually show that this thing can be a phone. And that, it just made me crazy that we had that perception, I mean, I-- It was rightfully there at the beginning of the project, but I felt like as we were meeting these milestones, I wanted to make sure that people understood that we're not a science project anymore, that this is for real. And so, so we did, we worked very hard to kind of deliver core functionality sooner than it was available on P1. And so one of the big, one of the big highlights was SMS. You know, I really wanted send and receive SMS working before the P1 team could deliver it. And so, it was either right before Christmas 2005 or right after Christmas 2005 that we actually managed to demo it to Steve. But it was just part of, I just wanted to have just more and more of these demos and have them running on real hardware, not running on a Mac as much as we could, as soon as possible to show that this thing is real and that we intend to ship it and we're not slowing down or we're not taking the foot off the gas just because we have another interim plan right now either.

Hsu: And so you did get that feature implemented before P1 did?

Ganatra: We did, yes. We did actually get SMS sending and receiving working before the P1 team and we were able to demo that as well. Now I'm not sure-- I, I don't know that that was the thing that killed P1.

Hsu: <laughs>

Ganatra: I don't think that it was. But, I just wanted to keep adding to that perception that, "Well, while you guys are still trying to figure out keyboard input for the P1, which was not a done deal, we're over here, you know, delivering core functionality."

Hsu: Right.

Ganatra: And so, if you're real-- we-- I wanted to keep the executive team and Steve as excited about P2 development as we possibly could, and what better way than showing new features and functionality on an aggressive schedule.

Hsu: Right. Were they thinking of-- I mean, you mentioned the keyboard. Were they thinking of shipping a physical keyboard at any point?

Ganatra: No, no. This was the using the click wheel--

Hsu: Oh.

Ganatra: To actually input text.

Hsu: Oh, my God. <laughs>

Ganatra: So the idea, yes. So the idea was to actually turn the click wheel, to pick the letter that you want, hit the center button to select. On to the next one. Click, send, click, you know, click, select, click, select, click, select.

Hsu: Oh, my God.

Ganatra: That type of thing. And also to have, I mean, and there were some of these--

Hsu: It's like going back to a rotary dial phone. <laughs>

Ganatra: In a sense, it was a rotary dial. So from my recollection in the meeting, to his credit, Phil Schiller thought it was a bad idea right from the beginning.

Hsu: <laughs>

Ganatra: You know, before we were even-- Before we even had the discussion about this P1 plan and schematics were pulled out and all that, already, Steve and, Steve Jobs and Phil Schiller were sort of arguing back and forth about whether a click wheel could be used for text input, and Phil just thought it was a terrible idea. But Steve just shut it down right away and we started having the meeting, you know. <laughs> So, so even from the beginning--

Hsu: So Steve wanted to actually consider it?

Ganatra: Oh, yeah.

Hsu: As an option.

Ganatra: Oh, absolutely.

Hsu: Huh.

Ganatra: He was completely on board with considering it as an option. And I think that out of that, actually, I think that may have been the first time that anybody had implemented the predictive—the-- where the keyboard actually predicted the word, the next word you were going to type in.

Hsu: Oh. Autocomplete.

Ganatra: Right. Like auto-- Like so if you type in the word there, T-H-E-R-E, then you would see an "is," you know, is the first thing. And then you select is and now "a" comes up, and you can select “a,” and--.

Hsu: Right.

Ganatra: That type of thing. I think that may have been first seen with the click wheel text input.

Hsu: Because it was so cumbersome, so you had to have--

Ganatra: Yes.

Hsu: A predictive text thing to--

Ganatra: Exactly.

Hsu: Help you out. <laughs>

Ganatra: Exactly. Exactly.

<laughter>

Ganatra: Yup.

END OF THE INTERVIEW

