Oral History of Nitin Ganatra – Part 2

Interviewed by:
Hansen Hsu

Recorded May 30, 2017
Mountain View, CA

CHM Reference number: X8186.2017

Hsu: The date is May 30th, 2017. I am Hansen Hsu, and we're back with Nitin Ganatra. So let me go back and sort of set the stage for where we are, where we left off last time. So, we had talked about the tensions between P1 and P2 teams. We were talking about when was the P1 project actually killed, and was the, I guess, the demo of SMS, was that a factor in that? That's exactly I think where we left off.

Ganatra: Mm-hm. Yeah. And I never—to be fair and to be clear, I never heard from anybody that it was the SMS demo that killed things, but, so it could just be pure coincidence that it was a week or two after that that it seemed like the P1 was sort of deemphasized and P2 was back to being the plan of record. So, I believe that was in early 2006 that that all happened, so shortly after the Christmas break thing and couple weeks after that. That's just my recollection right now and, I mean, as far as the team goes, we never—you know, just because the plan of record had changed to introduce this new milestone in between and we were a lot less involved with that, we had never slowed down our work anyway, and so when I had heard—and now I don't even remember where I heard—that P1 was now deemphasized, it's not like we sped back up again. <laughs> We were working at the same rate and sort of careening from demo and milestone to milestone and, you know, the work was just sort of building the whole time. As far as how that changed us, I mean, it certainly took the focus off of trying to compete with some other platform, as far as just demoing deliverables and things like that, but as far as the velocity of work, it never—it didn't slow down, so it didn't pick back up again or anything like that.

Hsu: Right.

Ganatra: Yeah.

Hsu: So, my next question is maybe switching gears a little bit. So, Scott Herz actually told me that some of the early code, or maybe framework stuff, that the team did was actually written in C rather than Objective-C, so is that something you might—could you talk a little bit about that?

Ganatra: Yeah, yeah, yeah. So, there were two reasons why some core components were written in C instead of in Objective-C. The two big reasons were we didn't completely know what our RAM requirements were going to be and even our image, like, the size of our image file that we could have allocated to the OS, you know, as far as storage goes, or anything like that, for the actual OS image, and so—let's see. I'm sorry. Did I—was that the first reason or the—

<laughter>

Ganatra: So, there were two reasons. One of them was the image size we weren't really sure about. The other thing was that when this P1 work had started, we also wanted to kind of use that as an opportunity to continue work and continue sort of proving out our core components as well as much as we could and so in the same way…. In the same way that—you know, just because we had this new P1 milestone, we really wanted to make sure that the P1 team didn't have to go and rewrite a brand-new Address Book, or a brand-new way of searching contacts, or importing Vcards, or sort of all the traditional machinery around something, you know, like the core work that goes into managing a contacts list. And so, what we chose to do was work with the P1 team and agreed to just deliver something that was in C99, just sort of C99 compliant, a vanilla C code that was in Address Book framework or an Address Book library that they could then recompile on the P1 platform and use as their core Address Book. And then, at the same time, we're using the same thing on P2 so that once P1 ships we will have a core Address Book engine that we know has gone through one GM release so we have some confidence in those pieces, and the more we could build up, you know, the more we could sort of flush out the development and frontload a lot of the development on these core components and make it so that we can trust them later, then it just gives us more freedom and flexibility later. So, those were the two reasons we really wrote some of these components in C. The big one that comes to mind is Address Book. I think that we shared some, like, a SQLite, sort of an object persistence layer as well, that the Address Book was built on top of. Beyond that, I don't recall if there were other pieces that we wrote in C, but that was the main idea behind it. Now, you know, having said that, we were still unsure if even that was going to be enough, you know, to write, <laughs> to write this really lean core, you know, code just in C and then have other frameworks written in Objective-C or have some thin bindings around it. It was still unclear to me if that was going to be enough, you know, but at least we were—you know, even if our RAM requirements and our image size requirements went way up, like, they, thankfully they did later, <laughs> we would still be able to front load some of the work. So, there was, you know, there were still advantages to delivering in C those core components.

Hsu: Right.

Ganatra: Does that make sense?

Hsu: Yeah, yeah, yeah. So, it's sort of like a way to, whether or not P1 or P2, whichever one of those was the decision, you would have something that you could share with them, that they could use some—

Ganatra: Right.

Hsu: —of your code still and they could, you know—you'd have something that they wouldn't have to re-implement.

Ganatra: Exactly.

Hsu: And so, this was a way to, like, have more code that was sharable, to have it just pure C.

Ganatra: Yep.

Hsu: Yeah.

Ganatra: Yeah, yeah, exactly. More sharable. I don't know why I didn't say sharable in the five-minute description I gave, but yes, absolutely.

Hsu: <laughs> I think Scott told me that some of it was intended for the web team too. Is that true?

Ganatra: Yeah. There was—what was the web team? I think that there was some event handling code that we—you know, some very low-level event handling and event maintenance, sort of user events. Some of that code was also written in pure C. There may have been some other pieces as well, but yeah. But it was really very early on we heard—you know, we didn't know what our requirements were as far as—you know, we knew ARM-class [processor] and we knew some details about the graphics parts, you know, parts that we were going to use, and we had some other broad understanding about what was going to be there: Bluetooth and Wi-Fi and a speaker that you could play through and things like that. But, really, as far as the RAM goes and the strict, you know, the speed of the processor itself, a lot of that was still up in the air. And so, part of it was, you know, assume the worst and hope for the best, you know, and so part of assuming the worst is just, well, what is the smallest requirements that we may be asked to run in and how can we make everything as lean and tight and small as we possibly can and yet have all the full, you know, the functionality that we were seeing in these demos? You know, how could you, how can you square both of those? And so, a lot of it just came down to efficiency of the implementation and efficiency of the, you know, even code size and, as you know, there are lots of aspects to performance that can be tweaked and that need to be managed. And so, we were looking at everything that we could just because we didn't know what those constraints were going to be.

Hsu: Yeah, hm. Earlier you mentioned you'd had a little SQL-like persistence framework, so that was not Core Data, right, so—because Core Data came later, so that was just something small that you guys did?

Ganatra: Correct, correct. Yeah. Core Data came later. Actually, timewise, I'm not sure if Core Data—

Hsu: Core Data on iOS came later at least.

Ganatra: Core Data—yes. Absolutely. Core Data on iOS came later.

Hsu: But it was already existing on OS X?

Ganatra: Right. Right.

Hsu: Yeah.

Ganatra: And there was already a fair amount of work and a fair bit of understanding of how SQLite worked. And SQLite was, at the time, was this relatively new open-source library that you could use to manage SQL databases or create or what have you. And it was very small and very efficient and worked really, really well, but then, you know, but it's still just straight SQL and so, yeah. We wanted something a little bit higher level for—you know, to address most of how people want to use a database, and that was the—yes. So, it was not Core Data, but it was just a very light wrapper that had some nice behavior that Core Data also shared later, like being able to fault in fields. To be able to create a large number of objects but have them be sparsely populated, as far as the data fields themselves, but then, as you go and query those fields, they can be bulk-loaded in from the database and have those objects populated in a very efficient way, sort of as needed instead of doing everything up front or doing anything too expensive. So, it did—yeah. It had some really nice features and we used it for many releases after that too.

Hsu: Hm, okay. Cool. <laughs>

Ganatra: Yeah.

Hsu: Let's see. Another one. So, let's talk about the "rule of three."

Ganatra: Uh-huh.

Hsu: You mentioned that earlier. So, what was it? What was the motivation for—what is the rule of three and what is the motivation for that?

<00:11:15>

Ganatra: Okay, okay. So, the rule of three was if you're going to share—I'll just start with the basic definition and kind of give the motivation after that. So, the definition was, if you have a piece of code somewhere, if you need to implement some functionality. You're Jane Engineer and you need to go and implement something, you may know that that piece of functionality is already implemented in another app or in another project close by. And so, the rule of three was you're allowed to go and duplicate that code, that chunk of code, duplicate it, as long as you're not the third client to go and duplicate that code. As soon as somebody, as soon as a third client needs to come and duplicate that code, then we need to really have a discussion about whether that code belongs in a framework or not. If, you know, in other words, should we just take—instead of having three different copies of that code throughout the code base, which then need to be, possibly need to be bug fixed and if we add new features or functionality we now have to add things in three places, if, as soon as you have a third place like that, it becomes a lot more desirable to instead not have three copies of that code but put that one, put one version of that code in a shared location and have all three clients use that shared code instead. Now, the motivation for, you know—you might think, "Well, why ever move code?" You know, "Why have this rule of three and why not just duplicate the code whenever you need to?" I mean, the obvious, maybe obvious answer for reason why that wouldn't be practical is just because now you have n-copies of functionality all throughout the system. Not only does that make your system larger because you have by definition of what we described, copies of this code everywhere, but it also means that you have a management, a maintenance nightmare. If you have a bug fix, you may think of the two places where you have code copy that does that thing but you haven't thought about the other three. So that's the reason why we don't want copied code to proliferate throughout the project, right? Is reasons like that. However, it is also, in my opinion, and I've seen this in other projects, it's possible to overshare as well. To go and create dependencies where you may not need to, and a lot of times it is just more practical. In spite of all those reasons why you don't want duplicated code, trying to figure out where, you know, reason about where things should, you know, where all the fixes should go and things like that. Sometimes it is better to just not have that additional dependency and to just, if it's just a little bit of duplicated code it's, you know, it's probably going to be okay and especially if it's been modified enough or it operates under different enough, a different enough environment or under different enough constraints that you really can't call it duplicated code anymore. It may have started duplicated, but it's been largely modified, so… But the other side of that, the other downside is, if you have code, if you decide that everything that you ever write could possibly be used by somebody else, and you want to go and, <laughs>—you know, the very first time I go to use a navigation arrow or something like that. Well, that's a really bad example because that is something that everybody uses, but…

Hsu: <laughs>

Ganatra: But the first time… <laughs> The first time I go and create a whosie-whatsit music widget, I created this great thing, everybody may want to use this. So, I'm going to go and drop it into this shared framework. Now, if everybody goes through and starts thinking that way, pretty soon you're in a situation where you have these shared frameworks that, because they're shared by everyone, they've now become larger than they need to be. Initialization time for those frameworks may be longer than it needs to be as well, so you can actually start to impact efficiency if you have too much shared code as well. And so, the rule of three was kind of a nice way to sort of strike a balance between just having a broad understanding that we don't want to introduce dependencies or we don't want to pay the cost for creating our—for having our frameworks be too large. However, at the same time we need to be smart about when we're using this code, we don't want to trip ourselves up by having too many copies in any different place, and so the rule of three was kind of a nice way to address sort of both issues on both sides and still have a lean and mean code base that people can work in and be efficient in. A lot of what we thought about early on was, and a lot of the decisions that we made, were in, you know, sort of in anticipation or were—the bigger requirement was that engineers need to be as productive as possible and so, you know, and that bubbled through so many different choices. You know, the language we use. We knew that engineers were already comfortable. There was a small group of engineers in Apple, relative to the whole world of engineers who don't know a thing about Objective-C, but Apple—you know, we had a good choice of engineers to choose from, a good selection of engineers to choose from to help with the iOS development. They were fluent in Objective-C, so Objective-C seemed like a reasonable choice. Those same engineers were comfortable with how AppKit and Foundation worked, as far as the patterns that you typically see, and so when we were creating UIKit it was very deliberately created to [be] sort of in the style of AppKit, to make it so that people could be productive for as long as possible, or as productive as possible. And so, that was the same thing with the rule of three was just in anticipation of keeping people productive, yet not making a mess of the system and trying to keep things performing as quickly as possible, you know, we had these little rules like that.

Hsu: Right. Yeah. So, we're talking about essentially pushing things down into the UIKit. Maybe let's go talk about the sort of how the UIKit was built maybe, and, well, first of all, who came up with the name UIKit, and then second of all, how did things start to become part of that framework? Which design patterns did you choose to bring over from AppKit? Just the process of designing what, or not designing—was it organic or did it just come about naturally?

Ganatra: Mm-hm. Yeah. I think that certainly a fair bit of it was organic. Scott Herz, I'm 99 percent sure he's the person who came up with the name, UIKit, itself—and it was, it seemed like, a fine name at the—

<laughter>

Ganatra: It seemed like a fine choice at the time. You know, we certainly didn't want to clash with anything that was already on the system, so we didn't want to call it the iPhone—well, I mean, we could've called it iPhone AppKit or iPhone UI. I don't know. There were other ways that I guess we could've made it so that we weren't completely just mimicking what we saw on the Mac OS side, but, at the same time, we also wanted to kind of give it a— you know, sort of treat it as its own thing and sort of look at it as its own platform and as its own, yeah, just its own world and ecosystem in the future, even though we weren't necessarily thinking about it that way at the time. But there was definitely a strong emphasis by Scott Forstall, as well, to really, "Let's make sure that we treat this thing—we do what's best for the iPhone itself," and not necessarily what's best for a team of engineers who work within Apple who may be <laughs> working on this or who may have to reason about this thing in the future. Like, really it was, you know, Scott said sort of repeatedly, "Let's treat this thing like a startup." And I think not only did he mean the velocity of a startup and the commitment of the engineers as a startup and the—

Hsu: This is Forstall?

Ganatra: Yeah, Scott Forstall. But also, "Let's think about what's right for this product and let's, you know, even if it means that our life is going to be more difficult as we fold this back in with Apple, we need to make sure that we're taking, we're putting our best possible effort into creating the best possible product here." And if that means that we're going to have a more complicated relationship with trying to fold this back in later, that's fine. We'll apologize for that. I'd rather apologize for that later than for having a less great product up front. You know, let's make the product as great as we can and put all of our energy into that and we'll worry about the rest later, and so yeah. So that's where UI, you know, the name UIKit came. Some things were obvious candidates to go into UIKit, like, for example, keyboards. Not everybody's going to go off and create their own keyboard, or nor do we want them to. <laughs> We want to have—you know, for any system you want to have some UI widgets or affordances that are familiar to the user so that when they come into a brand-new app that they've never used, they might at least have a clue of how to use the thing just by looking at it. And so, those types of shared components, those were obvious candidates to go into UIKit early on. As far as the implementation goes, delegation was a big thing that we used. You know, protocols. Even the style of the names that were chosen for classes, and method names, and wherever we could follow a pattern that was previously established by Foundation or AppKit, that was what we followed, because we weren't looking to—you know, as much as I was talking about doing what was absolutely the best possible thing for the phone, it's arguable that reimagining collection classes is the best possible thing for—you know, it's—I've seen other platforms go by or other OS efforts where these types of things are just created from scratch repeatedly and it really doesn't amount to much as far as whether that platform was viable later, or successful later, or things like that. It just seemed like a wasted effort, and so with—I mean, Foundation is a fantastic framework for the things, the low-level things that Foundation handles, and so why would we ever change that? And now that we're using Foundation, we're using CF, we wanted to also create a system that was somewhat consistent with the patterns that were in those frameworks, in addition to—you know, and we knew that if we kept that consistency we would keep high levels of productivity for engineers as well, that they'd be looking for, "Well, what is the UIKit equivalent to the delegate that finds out when a, you know, something is going away?" You know? They would be speaking, those engineers would be speaking a similar language and they'd be 80 percent of the way there to understanding how to use the iOS system, even though they had never seen a line of code in iOS before, and that was definitely seen as being very valuable too.

Hsu: Right. And one thing that really strikes me about UIKit is the heavy use of Model-View-Controller, almost even more so in a way than the AppKit, because you actually have classes named this Controller, View-Controller, you know, this-Controller, that-Controller, TableViewController. They're everywhere, right? And [with] the AppKit, sort of people just wrote their own controllers. There wasn't a controller class that you inherited from. Was that something that was, you know, explicit, you guys wanted to really push that forward, that way of doing things?

Ganatra: So that, interestingly—so view controllers, I think, were introduced—I'm not even sure. Were they in 2.0 in the first SDK?

Hsu: Oh, so that was later when the SDK came out?

Ganatra: It was later.

Hsu: Okay.

Ganatra: It was definitely later, and that was—and to be clear, I mean, <laughs> the engineers who were working on those early versions of the apps, you know, I heard from a number of engineers. You know, the nice thing is you never have to guess about what an engineer is, what's causing an engineer some pain or suffering or anything because most of them will just tell you right away.

<laughter>

Ganatra: You know, maybe unsolicited. But I definitely heard from a number of engineers that, "Hey, we're"—you know, "Some patterns are starting to emerge," as far as when I hit the Nav bar button, you know, I need to get rid of these views and I need to bring in this other set of views and I have to write all of the code to manage what's going where and these screens that, you know, these 'screenfuls' of widgets. Every single app had to manage those things on their own, and it was certainly, you know, it was something that was identified as being sort of a cumbersome pattern that a lot of the 1.0 apps had to follow, and so each one of the apps created sort of something that was sort of kind of like view controllers, right? You know, was kind of an easy way to, or an easier way, to manage these screenfuls of widgets and transitioning between them, and so it was only later, I think it was, I don't know if you remember, Alex Aybes, from Address Book. Formerly from Address Book.

Hsu: Okay. No, I don't think I ever knew him.

Ganatra: Okay. But yeah. So, Alex Aybes and I think Evan Doll were the two authors, and Chuck Pisula, I know he contributed as well to that, to the view controller effort, but it certainly, it took many, many, <laughs> many months of work and sort of a lot of rumination before everybody could sort of agree on, you know, even the name, "ViewController," and because it is a little bit loaded, right? The name itself, it sort of implies that, well, you're tying a view to some behaviors. But, you know, if you're in MVC, is that really, I mean, it is the, you know, it's a great—you know, it is, in my mind, the right abstraction for how to manage flow within an iPhone app, and iOS app, but at the same time it's, as you know, when you're going through and creating these things in a vacuum and trying to satisfy a group of clients and sort of anticipating, satisfying all of their current needs and anticipating future ones, it can be a little tricky to get that all right, but yeah. Eventually view controllers came around and everybody sort of threw away their own way of doing things and moved to it.

Hsu: Right. And was sort of that model of every time you transition to a deeper screen sort of pushing that down onto this stack, was that something that, again, was one of these patterns that kept coming up over again, that—

Ganatra: Yeah, yeah.

Hsu: —that they just ended up putting into the framework later on?

Ganatra: Exactly. Exactly. I mean, it wasn't, you know, that was certainly a case where we didn't follow rule of three just because the code that you would have to manage as far as pushing into a shared framework was everybody had sort of created their own, such a specialized version, that it really was just tailored to their own needs, and so no single implementation at that time was really suitable to be, you know, <claps> "Okay. This is a View Controller," and have everybody go and rename their classes and throw out code and it wasn't a straightforward transition to be able to do that, but yeah. I mean, that was something that we noticed as far as the UI pattern goes and it had to—you know, I think we have to give credit to the iPod, you know, and the click wheel and just the idea of selecting and driving the navigation kind of from—what is it? From left to right, <laughs> you know. Sort of moving the—you know, as you go in details deeper and deeper, moving along, but yet still having a way to get out. I mean, yeah. It was something that we had noticed early on in the early designs of iPhone, the iPhone UI, but it really, I think, took going through and creating a number of apps, because before really understanding that this was a pattern that was emerging that a lot of different apps could use, because at least with the very first designs that we were seeing, yes. I mean, you could look at it and say, "Aha. Well, we probably need this sort of way of diving into details and managing this hierarchy of views, of screenfuls of data," but even at that time, I felt like it was too early to go in and start reasoning about what would be an abstraction that we could use to encompass all of this. It was—you know, early on you kind of want the design team to just kind of go wild and, you know, maybe we'll end up with something different and even better than sort of this detail pushing its screenful of content across your screen, and so there's a little bit of—you know, it's not always a slam dunk to just look at a system, at a couple of different views of a system, and then start trying to reason about an all-encompassing hierarchy of, or architecture for how all software will be developed. You don't want to do that too early in the process, because you start, you know, a lot of it can just be wasted work or a wasted effort or you end up at the very end with something that looks nothing like what you anticipated at the beginning and it's just kind of a hot mess. So, I think that that's part of sort of the art of engineering as well as knowing when to actually go in. You know, when is the right time to go in and create a different abstraction and a shared way of—you know, a shared set of facilities that can be used by multiple clients. You really have to kind of understand what those clients are doing and what their needs are before you can go and create something that's suitable for everyone, and how can you do that if you haven't lived through those hard experiences for a little while up front, you know? If you haven't done that, then you're basically just guessing at what the functionality is that you want and your guesses are going to be bad in some cases.

Hsu: Right. So, a lot of it, it sounds a lot like the process was simply, rather than starting out and designing a framework, you guys are just working on the apps. Like, “Let’s just implement the designs and bang out these apps and then if it happens to be rule of three, there’s three different things that are all sharing similar code, then let’s push it into a shared framework,” and then gradually over time the sort of collection of shared things sort of builds up and you have this sort of overall thing that you guys call UIKit.

Ganatra: Yeah.

Hsu: Sort of like that?

Ganatra: That sounds like a very, very accurate <laughs> description of how the development went. You know, it was—and I do think that there are—and it’s not like it didn’t occur to anybody to go create an all-encompassing system for transitioning between screenfuls of widgets, right? I mean, that was something that people had been thinking about early on as well, but when you’re just looking at the first five or six views in the iPhone UI, it’s hard to anticipate what those, what this all-encompassing set of classes should do, and a lot of times it is just better to kind of, yeah, just to straight to implementing what you know you need to implement and let the process of actually writing the code and the process of developing this sort of lead you to where you’re going to go. If you try to create this architecture too early on, you can kind of stifle the ability to go in different directions or the ability for, yeah, for the HI team to kind of go and go hog wild, you know, like they did in a couple of cases on the iPad UI, you know? And so, it’s just better to kind of just let it be organic for a while but understand that there’s a plan there. You know, that yes. The plan right now is to be organic and to let it sort of go for a little, let development go for a while so that we’re—because you’re gathering up requirements the whole time and you’re gathering an understanding of how these different apps need to work and all of the little details that this grand shared architecture for transitioning between screenfuls needs all the things that it needs to eventually do.

Hsu: Right. Because there’s the, like, there’s the worry that—or what you don’t want to have happen is that the—you’ve overdesigned this framework and then the infrastructure then starts to dictate what you can and cannot do in the UI and that’s the exact opposite that you want to have happen.

Ganatra: Exactly, exactly. Right, right. That was our—you know, the sort of the meta goal and I’m not even sure that we really ever discussed it, but really one of the goals was to sort of let the HI team do, come up with, the best possible design and not bog down the HI team with discussion about number of colors on the display or how much RAM we’re going to have or how much, you know, this, that—you know, like, you know, obviously as an engineer you’re thinking about those constraints all the time and you’re thinking, “Holy crap, am I going to be able to implement what, you know, this really cool thing that the HI team came up with?” But really you want to, you don’t want to start to burden the HI team with these constraints early on because then the best thing that they’re going to ever come up with is a design that they themselves, the HI team themselves, anticipate will work given all the constraints that were given. You've now put the HI team in a situation where they're now trying to guess at whether their design is still really cool enough to be compelling to a customer but still something that can be implemented. It's really sort of, it's better to just sort of stand aside and let HI come up with the best possible designs, and especially that, you know, the HI team that Apple has, and had at that time. I mean, you really just want to kind of get out of their way and come up with the coolest thing possible and then we'll figure out if we need to, you know, if we need to implement something that's 90 percent as cool but is possible, <laughs>on an ARM-embedded device or what have you. We can have that discussion later, but you come up with the best design, the engineering team will come up with the best implementation that we can of that design, and if there isn't, you know, a complete match between the design and what can be implemented, by then we'll have a better understanding of what the details are, where it's difficult to implement things, and we'll have, on the engineering side, we'll have specific recommendations for, "Well, yes. You said that this thing could be an unlimited length table, but what if we made it a thousand cells instead?" You can have more, you can have very, you know, sort of specific designs that address the specific problems you're running into in engineering and minimize the impact on the great work that the design team's done. So, does that make sense?

Hsu: Yeah, yeah, that makes a lot of sense. Sure. You know, that brings up another thing that I've been wondering about. Certain technologies that had come over from, that were available on OS X, became pretty important in actually implementing the iPhone OS. You mentioned Foundation earlier. I'm thinking about Core Animation, like, the LayerKit. How critical was that technology?

Ganatra: It's completely underrated today as far as how important it was to have that component in there and that was—

Hsu: Mm-hm. Could you describe a little bit for us what that is?

Ganatra: So, Core Animation was formerly called LayerKit and the idea was that we have graphics processors on computers and on phones that are powerful enough, such that when it comes time to actually composite a given image, whether that's rendered, like, from a PDF that you read off of disk or if that's rendered by, you know, programmatically through methods that an engineer has written, but once you actually have a view that is completely drawn out, that compositing that view onto other views or layering them with other views, is nearly free. It's a very low-cost operation and that's all possible because of the way LayerKit worked and made it so that, you know, instead of creating views for things you actually created layers. I mean, layers and views are, you know, in a lot of ways they're conceptually very similar, you know? But the nice thing about a layer was that you could actually composite it in with other layers, a.k. other views, and that composite step, because it was so inexpensive, you could do things like have smooth animations or you could have smooth transitions as part of that compositing step, which led to kind of a richer feel of what's going on in the UI. So fading something in or out of the screen just became a matter of drawing it, you know, either loading something from disk or drawing it, and then the actual compositing step was where you could actually say, "Well, apply this Alpha in an animated way so that it looks like this view is either coming onto the screen or fading in or fading out, but it's nearly free as far as the CPU goes." So, all of the work is done on the graphics part and the graphics part is just tuned to be able to do this compositing very, very efficiently. And so, what you end up with is a system where you can have these rich animations that are happening but there's a minimal hit on the main CPU itself. A lot of the rich animations and the smooth, fluid scrolling that you see can all be handled by this other very capable graphics part, freeing up the CPU, but it can also be—the way that you program that or the way that you use that system, is at the same time, it feels very familiar and very comfortable to developers too. So, it was critical to the success of the iPhone and to the UI itself and—yeah. Yeah, it was, I mean, it was John Harper who did that work and Scott Herz and John Harper worked together early on to kind of create the early versions of the LayerKit and Scott sort of wrote early versions of the, you know, like the Address Book UI or different things to kind of prove out whether this, you know, whether this works as well as everybody would hope it would.

Hsu: Right. So then, I mean, because I remember at the time that you could do things like that in, directly in OpenGL, but there was this sort of cognitive load to learning OpenGL, right, and—

Ganatra: Yeah.

Hsu: And so, LayerKit was a way to have a much nicer, much easier to learn abstraction so that you wouldn't have to, you know, go down to the GPU and write shader code or whatever?

Ganatra: Right.

Hsu: <laughs>

Ganatra: Yeah. Yeah. You know, and everything that you did when you were using LayerKit, it was all in terms of just Objective-C. You know, it really just felt like you were writing AppKit drawing code when you're actually—when you're actually drawing into a layer, but then, the animations were all, you know, were where a lot of the magic happened and that was a very easy thing to sort of manage as well. You could just plug in the duration, you could sort of, you know, plug in the two layers that you're sort of transitioning between. I mean, it was a very—you never had to drop into this alternate bizarro language of either OpenGL or something more hardware specific, you know, as far as programming the GPU goes. You could just use everything largely how you use, as you, how you do any drawing, either an AppKit or an iOS, yet you get all of these benefits for it as well and you can control these animations and things like that. So, yeah. I mean, I think that that was a big part of it too was, yes, technically a lot of those things were possible on Mac OS and on other systems, but if they're not easy enough to use and if they're not available, if it's not available to a developer in an easy to maintain and easy to reason about in a manageable way, then developers are going to start to weigh that, and after a while, you know, you have engineers saying, "Well, gosh, do you really—<laughs> are you sure you want an animation there? Why don't we just pop from this thing to that thing instead?" Just because it's easier and I don't have to write a bunch of OpenGL gobbledygook. You know, I can just, I can crank that out in no time. <laughs> So, you know, I don't think that we would've had the system that we have today, just like on Mac OS, yes. It was technically possible to do a lot of the animations, but just like on Mac OS, a lot of those animations weren't built into the system in a way that was reusable to a lot of the apps. Mac OS doesn't have the amount of animation or the transitions, especially back then, that iOS does and I think that just by having the LayerKit in the toolkit and have it being this easy thing to use and, because it was easy enough, to use, it was so easy to use that you could sort of physically, you could look at a design in an iOS app and you could imagine to yourself, "How am I going to write this code?" you know, the animation code, using just UIKit or LayerKit calls, and you're like 90 percent of the way to what you imagined you would have to write. And so, that's a very powerful thing because then it's just, it's so easy to add animations that you can create a really great system with minimal effort.

Hsu: Yeah. It's like having this animation framework just there, this toolkit there, meant that practically anything that the HI team came up with you were ready to implement. Like, it wasn't like you had to reinvent the wheel every time, you know. It's like—

Ganatra: Right.

Hsu: —it was a very easy match from HI designs to actually getting it implemented.

Ganatra: Right, right, exactly. Exactly. And the thing that you were operating on were the actual views or the layers that you yourself had already created. You know, a lot of times when you're doing this switch into it, something like OpenGL, you have to sort of take the state of the world as you understand it in AppKit right now and replicate the state of the world in, you know, in OpenGL speak and then do the, you know, perform the magic that you want to in OpenGL and then copy the state of the world back, you know? And that can just be very cumbersome and things aren't a perfect match all the time and then you have to figure out what to do. So, the fact that this was all literally just built right into UIKit, you could do animateWithDuration, you could—you know, just these straightforward methods were available to do whatever you want. Yeah. It made all the difference in the world to—you know, just for ourselves. You know, of course, initially we weren't really thinking about an SDK. You know, all we were thinking was we know that we're going to have to, you know, this is a—the UI is so animated on this system that we need to make it easy to have these animations and that was the justification early on. It was just, again, it was just this very practical concern that, "Okay. We have animations everywhere. We need to make it as easy to use because it's going to impact productivity if we don't."

Hsu: Mm-hm. Yeah. Huh. Have another really highly technical question that it's not initially on my list but I just thought of it. <laughs>

Ganatra: Uh-huh.

<00:45:35>

Hsu: So, around that time, maybe a little bit later, I don't know exactly the timing. I think maybe it was 2006 that Objective-C 2.0 came out that had support for garbage collection in Objective-C, and there was not a lot of take-up on the Mac OS X side because all the legacy apps were still using manual retain and release, and they had to be completely—they had to be rewritten to take advantage of garbage collection, but with the phone you're on a completely new—you're starting from scratch, right? So, was there any thought about using that from the get-go or was it always, "No, this is not something that we think is useful for the phone?"

Ganatra: I think—Gosh, that's a really good question because I remember having to deal with the garbage collection functionality coming into Mac OS, but I think that was in the, even in the Panther time frame, definitely in the Tiger or I'm sorry, Mac OS 10.4 time frame. I know that there was a lot of work that went into just taking Apple's low-level frameworks like the Address Book. You know, sort of the core Address Book framework and the main engine behind the Mail app and converting those things to use garbage collection. I know we did a lot of work on that as part of the overall effort to kind of, to have a complete garbage collected system, but, for some reason, I don't know if it was already deemphasized or if we had already made a decision before that, but we, I don't recall, having to—we did not worry about garbage collection.

<laughter>

Ganatra: I think for—it was either that the writing was on the wall that maybe this wasn't the right implementation or the right way to implement garbage collection on a Mac OS system and so, therefore, we're not going to push other clients in this direction. It could've been that or it could've been that, "Hey, we're operating all on our own and we're going to barely have enough RAM to do the things that we're actively managing," you know? "Why do you think we would have enough RAM to have a garbage collected system?" So, in either case, I remember it not being a big concern <laughs> in early iPhone OS development.

Hsu: Right. Because you're so concerned about performance and limited small memory footprint that it's like—

Ganatra: Right.

Hsu: It seems like a, maybe a profligate—

Ganatra: <laughs>

Hsu: I don't know. Like a waste of resources <laughs> to do that or was it…

Ganatra: Yeah, yeah. It was kind of a—I mean, in a lot of ways garbage collection is, exists, to make it so that the programmer creates fewer errors in their system, but it's really not—you know, and I know there are people who say that, you know, "Well, and garbage collected systems in theory can be faster." Well, I've never seen those cases, so…

Hsu: <laughs>

Ganatra: You know, that was mostly directed at Bertrand [Serlet]. Hi, Bertrand. You know, if you ever see, if you ever watch this. But he would argue that occasionally but I think he even came around to the later ARC[Automatic Reference Counting]-based systems instead, but yeah. We just never—I mean, we—because early on we barely knew how much RAM we were going to have available, such that it was dictating even our language choices. You know, I think that garbage collection was just something that was just sort of off the table. Like, "We'll worry about that when Mac OS has that just humming along perfectly, then we'll—and we have enough RAM that we can afford to actually have a heap large enough to have garbage collection work in a practical way—then maybe we'll think about it," but, you know, we never got to that. <laughs> Never got to that point. I think, as you know, we moved to this technology called Automatic Reference Counting instead and that's, you know, and I love it. I mean, I think it's a great technology. It's worked great. It does, it addresses a lot of the concerns that garbage collection had and yet it doesn't incur the performance hit. So, you know, it seemed like we landed in the right place, and thankfully we didn't have to, you know, from what I recall we didn't have to waste a lot of time on sort of the interim garbage collection solution, you know, from the early 2000s.

Hsu: Right.

<laughter>

Hsu: One thing that came out of that, well, this is sort of a slightly related thing. So, I interviewed Blaine Garst last year and I was asking him specifically about the atomic/non-atomic property for—yeah. So—

Ganatra: Objective-C properties. The—

Hsu: Yeah, the Objective-C property on—

Ganatra: Yep.

Hsu: And [I] had done a tiny amount of iOS programming, so I was always told that, “Okay. We always use non-atomic,” and it was like, “Hm, why is that?” And Blaine was like, “Well, the default is atomic.” [I wondered] “Why do we have to explicitly say everything is non-atomic when, you know, why is the default that way [atomic?]” and he said that, well, when they designed it for Objective-C 2.0, the default was atomic because the intention was that everything would be garbage collected, and in the garbage collected scenario you wanted everything to be atomic. But then when you guys were doing iOS, you weren’t doing that, and also there wasn’t—there was some concern about atomic not being performant enough and so that’s why you guys did everything non-atomic. So, is that the case or was there any thought about which one you guys should be using?

Ganatra: There was definitely thought about what we should be using, and, from what I recall, I think we landed on using non-atomic just because we didn’t want the locking and unlocking overhead, you know? We didn’t want the—you know, even though it’s funny that you say that. I always wondered, you know, <laughs> why that default was chosen. I figured there was some thought that someday object synchronization is going to be close to zero cycles and so if the only reason people aren’t doing this today is performance and if we address that performance concern, then that should just go away and really atomic should be the default. But it sounds like there was, you know, there was the garbage collection argument as well, which I can see, that makes sense too. But definitely for us, we just knew, you know, we were creating some a lean system and we knew how important that 60 frames a second number was or, later it turned out to be a higher number that we were chasing. And so, we had all these performance tasks running all the whole time and we had all these metrics that we were tracking and so the idea of taking a performance hit for something as commonly done as accessing a property on an object, it seemed like overkill, especially the way object synchronization is implemented today. So, down the road, I think our thinking was, “Well, yes. Down the road, once object synchronization is fast enough, then we too will switch to that <laughs> default as well, but, in the meantime, this is the world we live in.” And so, if that’s not the default, then we will just add it to everything, you know, add non-atomic to everything ourselves just so we don’t take that locking overhead.

Hsu: Right.

Ganatra: Yeah.

Hsu: Yeah. <laughs> I think that was—I think Blaine said that that was one of the cases where you guys being a secret project, it’s like there’s no communication going on between the compiler team and <laughs> your team, so the… <laughs>

Ganatra: Yeah.

Hsu: So, things kind of—decisions were made that maybe weren't optimal. <laughs>

<00:54:30>

Ganatra: Uh-huh. Well, you know, yeah. That's true. That's true, but at the time I would also question whether—you know, remember at, I mean, this was before the iPhone was released as well and so depending on who you asked it could either be a science project that never ships, to, you know, it could be another Cube, you know, the infamous [G4] Cube that Apple shipped, you know, in the early 2000s as well. It could just be a flop, in other words. You know, we could ship this thing and it could just not go anywhere, or it could just be seen as this luxury good that, you know, a very small number of people buy. So, I'm not a hundred percent sure that I buy that, you know, future decisions were hurt by secrecy because we didn't know what you were doing, because it also, in some cases, there were also, you know, other groups, and us included—like, none of us knew how successful this was going to be. None of us knew if this was going to be something that we would really be interested in rolling back into, you know, sort of all of the rest of Apple engineering, you know? Or is it going to be kind of like the Newton was, where, you know, remember, that was not a, it was not a big success and there was no effort—I mean, you know, there was some effort to improve the product over time but it's not like it changed the face of Apple like the iPhone did, you know? And so, yeah, it's easy to say now that we would've made, you know, decisions a little different, but I'm not so sure that that's the case because at that moment all the information that anybody had was that there was this group of overworked engineers <laughs> in IL-2 and, you know, they're working on something that looks pretty cool but there are a lot of questions about it too. So, you know, it's hard to say and, you know, if hindsight is correct there.

Hsu: Right. <laughs> Okay. So, another thing that you discussed in the podcast was deliberately not supporting things that users later clamored for later on. Like copy/paste, MMS, WAP. So, talk about like the general philosophy behind, you know, explicitly making those decisions, right? Like, "We're not going to do something," and why do we not want to do something?

Ganatra: Yeah. Okay. So, at the time, there were a lot of different—I mean, so we're talking 2005 here, right? And so, the internet had obviously, you know, it was, I guess, 10 years old roughly, as far as mainstream acceptance of it. Was about 10 years old, so that wasn't going to go anywhere. But up until that point, smartphones or what people called smartphones, they really operated on their own separate channels, you know? And there was largely voice and there was a little bit of, you know, there were some small provisions for being able to send and receive text messages, but really it wasn't completely connected to the open internet at that point. There weren't really phones out there with a full TCP stack on them, for example. Now, I know that people from Danger—there was a device called the Danger Hiptop that I think did have that and, you know, my buddies who worked on that are probably going to scream that I said that, but really, there wasn't really, the internet hadn't really made its way to phones yet. You know, the internet was largely this thing that affected computers and phones were these things that, you know, well, it was neat that now everybody had a cell phone in their pocket, but if you ever wanted to, you know, if you ever even thought to yourself that, "I want to go to www.apple.com," you were in for just a horrible experience, you know, and they were—it went from being almost non-existent to you wish you had never tried it because it exists but it's just so horrible that you don't even want to do that. So, in the meantime, there have been these technologies that attempted to bridge the gap between the internet world and phones, and a couple of them were WAP. You know, a couple of them were WAP and MMS. In my mind, MMS falls into that category too, which are these technologies that use some of the, you know, the infrastructure that's largely available on phones as they existed at that time, but they have very specific ways of connecting to the internet as well and being able to pull data over. In the case of MMS, pulling rich messages over. In the case of WAP, pulling a sort of a bizarro version of the open internet, of a webpage, and bringing that over to a phone. So, okay, again. So, this is all 2005. Internet is taking off on computers. Phones have these bolted on kind of technologies that sort of, kind of allow you to connect to the internet in some ways, but they're not really great, and we knew what we were creating in Apple, you know, which was this phone that had a full, real-deal TCP stack. It could connect things—you know, it could create secure connections over the open internet using Wi-Fi or using a cellular connection, just as if you were using a laptop that was on Wi-Fi at the same time as well. And so, we knew that we were going to be on the open internet and it was going to be a first-class client of the internet as well, and so the question was, there are these holdover, these technologies that had been created along the way. Do we want to adopt them? Do we really want to support these things? Because we're creating something that's a brand-new type of device here anyway, and sure we're, you know, by choosing to support things or not, you're kind of guessing as to, you know, whether or not you think something is going to be important, but, at the time, our guesses, and I fully support those, you know, at—supported those at the time too, were that, "No. These aren't--" you know, once we have a device that can get people on to the open internet, they're never going to want to use WAP and they're never going to want to use things like MMS because there are other ways of sharing photos and sharing data and, you know, small movies and things like that, and we don't need these holdover technologies anymore. We've got the real deal now. Why do you want the holdover when you've got the real thing? And so that was the argument for killing WAP support and MMS support. Sort of the two went hand-in-hand. Obviously, MMS <laughs> support came in later. You know, I think the thing we didn't anticipate was just the network effect of MMS and just the power that, you know, just how many people were already sending and receiving pictures using MMS was I think the thing that was, you know, the surprising detail to us that made it so that we had to revisit that, and scrambled and added MMS I think two releases later. I think that went [in] iOS 3.0. So that explains the MMS and WAP, why we said "No," and why we had to <laughs> scramble and change at least the MMS decision later on. WAP we still haven't. You know, I think we guessed right on that one. We guess way wrong on MMS, and then speaking of guessing wrong, you know, the other one was copy/paste. I think the thinking was that as great as we were going to make typing on an iPhone, it was still going to be a thing that you're probably not going to want to do for very, very long. You know, it's not going to be something that you're—you're not going to want to write a novel <laughs> on an iPhone and you're probably not going to want to compose a really long e-mail on it as well. And so, there was just some, you know, there were some questions—so I think that there was a little bit of a, you know, in my mind anyway, and maybe this isn't how Scott and Greg Christie and other people were thinking about it, but in my mind that was one of the things that made it so that, you know, if people aren't really going to want to type or author these long things on the phone, you know, do we really need something like copy/paste? And, you know, and along with copy/paste comes a lot of these other decisions too, like, “How do you implement text selection and how do you make that work really well everywhere?” And things like that. And so, just by mothballing copy/paste, you know, I think it saved us a whole lot of work that we weren’t really sure we were going to need to do anyway, because the truth of it was that we had this, you know, we had this largely touch-based phone where the screen is the input device and, you know, the keyboard works pretty well, but how are people going to use it? And even in our internal, the whole time that people were inside using the iPhone, even then it was, you know, the most typing that you did was kind of a, “Well, I’m going to send a text message.” Or I’ll send a probably little bit more detailed text message than what people used to send before, which was the letter “R” instead of the word “A-R-E” and, you know, the letter, “U,” instead of the word “Y-O-U.” You know, all of that kind of stuff. Well, now you can actually type, like, a real, you know, like a civilized human being on this thing instead <laughs> of sending this nonsense—you know, this gibberish that I’m supposed to parse and whatever, you know? So, I think that on that one we just guessed wrong just as far as understanding, you know, “How much are people going to be typing on these phones and how big a part of people’s daily work habits are these phones going to be?” Certainly, in my mind, I think it came as a big surprise to a lot of people at Apple to find out that for a lot of people, especially around the whole world, in developing countries, that a smartphone is their first computing device ever. And here we were, you know, we were rich people in Silicon Valley who’ve, you know, a lot of us grew up with computers since the time we were, you know, single digit age, you know? To imagine a world where this thing that we’re creating, this science project, is going to be the first time that people ever get on to the internet or ever have a computing device. You know, maybe that would’ve changed <laughs> some of these decisions if we knew what this thing was eventually going to turn into. But, you know, at the time it was just making <laughs> the best decisions we could and obviously, yeah, we guessed way wrong on copy/paste too and how much people were, how comfortable people were going to be with the keyboard and sending long passages of text and things like that.

<01:05:56>

Hsu: Huh. I wonder if copy/paste is maybe a—it’s maybe really a symptom of this larger thing, which is that, you know, the fact that on iOS, apps are all sort of their own walled garden, they’re all individual sandbox, they don’t talk to each other. At least initially, right? That people saw a need for—to be able to transfer data from one app to another, text from one app to another, and that was not something, you know, by design the iPhone is—it’s supposed to be everything is full-screen and when you’re in an app it’s, at least the experience is that there’s nothing else going on, that you’re just in that app and that’s a very different maybe paradigm, let’s say, from the desktop, which on the desktop you’re always having multiple apps running and you can switch windows between—back and forth and that’s sort of, in some ways, it’s maybe one of the criticisms that people have of iOS, is that apps are so walled off from each other. Could you maybe speak to the reasoning behind that going forward with that, that design?

Ganatra: Well, I think, yeah. Well, there are two things that I wanted to mention about that. One of them also that occurred to me was, as far as why we didn’t adopt copy/paste to begin with: So there was also, there was an understanding that we were engineers in the middle of a computer company who were developing this phone, right? And that throughout the development of the iPhone, there were different times when, you know, because we're all, everybody's working in a computer company, suggestions would come either from inside or from outside, or from wherever about features or functionality that a computer has that this phone should have, right? And so, in a way it would be very easy to just say, "Okay. Well, let's go and look at a computer," and we know that this is, you know, this is all the functionality that's on a computer and it's really useful on a computer, so we should just move that over entirely over to the phone as well. And the reason I bring this up is specifically with copy/paste itself, <01:08:45> I mean, the whole notion of cut/copy/paste and a clipboard, you know, to hold your edits, is very computer-centric. You know, it's very—I mean, that is something that, yes, it's very useful for input, for text input and for modifying text and things like that, but it is also, these are all very 'computer-y' concepts. You know, they're all things that came over from the computer and they were the best way to do it on a computer, but who's to say that they're the best way to do it on the phone. So, I know that for me, and I know that for other people on my team too, if people—it was never a great argument to say, to come up to anybody and say, "Well, this feature exists on the computer and it's useful, so that's why you should put it on the phone." Like, that was a—in my mind, that was a ridiculous reason to put something on the phone. That was like the last possible reason we should put something on the phone is because it was on the computer. And part of that is just because on the one hand you can just, you know, that can slippery slope into feature parity with, between a phone and a computer and is that really going to make a better product that you have all of the features that you have on a computer on your phone? It's not necessarily going to be a great product for everybody. Maybe for people who work in a computer company, that would be a fantastic phone to have, but for, you know, Joe Average user out there, they're not looking for computer parity. They're probably looking for the best possible communication device that they can have in their pocket with them, and that doesn't necessarily mean it comes from a computer. So I can say, <pauses> I could say with a fair amount of confidence, that very likely the first time I heard somebody say to me, "Well, you need cut, copy and paste on the phone," I was <snaps> probably allergic to it right away just because it was a, "Oh, you're asking for something that's 'computer-y,' that you find useful on a computer, to come over to the phone," and, I mean, in hindsight right now, yes. Obviously, you want something like that, right? <laughs> To be able to copy text over or to modify things or what have you. It's very useful on the phone too, but, in my mind, it wasn't even really necessarily the best solution that we have a clipboard and we have something, you know, these well-known commands for cut/copy/paste, be the thing that you use to edit text on the phone. Like, what if we did come up with something so much better instead of this hidden clipboard that, "Well, I copied something five minutes ago and now it's up to me to remember, 'Well, when I paste am I going to paste the thing that I remembered copying five minutes ago?' or is it going to paste something totally different?" I mean, there are problems with cut/copy/paste that people have all the time today that you just kind of live with and you know it's just the nature of how it works and it's just fine. You know, you get along with it and you do what you need to do, but just because it existed on the computer and somebody had the need to edit text on the phone, to me, it didn't necessarily mean that having copy/paste on the phone was the best possible way to edit text. And, in fact, I was probably allergic to, you know, the first time I had that suggestion it was, "Oh, you just want computer stuff on the phone." "No, we don't work that way. Let's do the best thing for the phone, and if it happens to exist on a computer as well then, 'Okay, that's fine.'" But just because something exists on there doesn't mean we want it over here either.

Hsu: Right.

Ganatra: So that was the first thing.

<laughter>

Hsu: Okay.

Ganatra: And now to answer your question. What was your question? <laughs> Sorry.

<laughter>

Hsu: It was about the idea that apps are stand-alone and they don’t talk to each other, right?

Ganatra: Yes.

Hsu: And they’re completely walled off from each other.

Ganatra: Mm-hm.

Hsu: Yeah.

Ganatra: Right.

Hsu: Which a lot of people kind of complain about nowadays.

Ganatra: Yeah.

Hsu: And actually, let me bring up something else too, which a lot of people now complain about, which actually goes directly what you’re speaking about, is not having a user-accessible file system for the phone.

Ganatra: Okay. Yeah.

Hsu: It's another similar thing—

Ganatra: Yes.

Hsu: —where people complain about that they have this experience on the computer, they have this freedom to control, and they don't have that on the phone and it's like, "Why don't I have that on the phone?" but it totally is, exactly falls into what you're just talking about.

Ganatra: Right, right. Absolutely. The exposing the file system is, well, we could do four hours on that.

Hsu: <laughs>

Ganatra: Just all on its own. You probably wouldn't want to. Let's do it over a beer sometime if you want.

Hsu: <laughs>

Ganatra: But as far as the full-screen experience goes, I mean, I think really that just came out of the designs that we saw. You know, it was just the earliest designs. You know, remember the phone screen itself was a lot smaller than when you buy an iPhone today, you don't have the three-and-a-half-inch screen that people had at the time. And so it was understood that you're not going to have—I mean, it's going to have a bigger screen than you could ever get on a phone before then, <laughs> but that was only true in 2007. As obviously when Android phones came out, they kind of showed that, well, people do actually want much larger screens and it's only once you have a much larger screen that you, to me anyway, I'm assuming everybody thinks like I do, but…

Hsu: <laughs>

Ganatra: You know, it's only when you have a larger screen that you start to think, "Well, what else can I put on the screen that can help improve my productivity?" Like, "Why do I need all of the space devoted to what I'm seeing here?" Well, I mean, early on you didn't have that space and so, you know, and it was also understood—in fact, not only did you not have that space, but there wasn't even enough space for all of the features and functionality that you want to have available in an app anyway, right? So, for example, things like settings: If you want to control settings for an app, there is no uniform control within an app itself that will take you to settings, right? You actually have to go out to the Home screen, go into the Settings app, and then you've got all of your settings there. Now, we know that, over time, people more and more have been adding settings straight into their apps and I think that that just follows because screens have been getting bigger, there's a little bit more room to spread things out and add things wherever you want to, but when you're talking about that original screen size, we were out of space. You know, there was no way <laughs> we were going to be able to do anything as far as bringing up, you know, adding, you know, not even—we couldn't even add all the widgets we wanted to, much less anticipate that people may want to do things similar to what they do on a computer, you know? And so, I think that that's part of what sort of drove the kind of the early decisions around, well, these things actually visually are walled off from each other. <laughs> You know? And we knew from a security perspective that we didn't want to create something like a shared system where your e-mail was in the same process as your contacts—or, you know, or, I mean, that's a bad example—but, you know, we didn't want too, too shared of a system either. We wanted to take advantage of this UNIX-based system that we had where processes could be killed off and resources could be cleaned up easily and things like that—so, you know, and sort of by default, you know, with a UNIX kind of system, you're just not sharing <laughs> between, you know, between two different processes, so…

Hsu: Right. But on the other hand, from a user perspective, users of UNIX systems can easily pipe the output of one process into another process, right? You don't have that kind of thing in iOS, right.

Ganatra: Right.

Hsu: It's much harder to get data from one app to another app.

Ganatra: Yeah, yeah. It's true.

Hsu: And even, you know, saving a file, you can only save your file into your little sandbox, and you can't save a file that multiple apps can access.

Ganatra: Right, right, right. Yeah. I mean, I think for a lot of the early decisions, a lot of it was sort of just by default, let's, you know, if things are partitioned off from each other and if we save things local to where an app is as far as its installed pieces and things like that, then it's just going to make management easier down the road. It's going to make it so that when we want to uninstall an app we know exactly where to go to uninstall. You know, it's a super-easy operation to be able to go and uninstall an app. If you all of a sudden have shared documents, or something like that, now you're in a position where the ownership of those things is not really clear. And, by the way, I think that that's one of the big problems with the—I think that that's one of the fundamental problems with exposing a file system as well, is once you have exposed the file system to the user, yes, now, that becomes an easy way for the user themselves to be able to share data, right? I mean, I think 95 out of the hundred reasons I've heard for why people want an exposed file system all amount to sharing data between where they authored that thing or edited it or whatever and where else they want to be able to do something else with it, right? And they don't want the program to get in the way of being able to take this document or blob of data and being able to share it. “I know what I’m doing. I’m the user. Get out of my way. You know, I know what’s going on here.” The only problem though is that once you’ve exposed the file system—well, two things: First of all, having an exposed file system like that is a very ‘computer-y’ thing, right? If you’ve never used computers yourself, it would never even occur to you to have an exposed file system, right? So, allergy trigger number one. But really, the big problem is that once you have an exposed file system and you’ve allowed the user to go in and share things or not, now the ownership of that individual file is really unclear. So, if you want to go in, if you go and delete Pages or something like that from Mac OS, it is most certainly not going to go around and delete all the documents associated with Pages, nor would you probably want it to, because the model for who owns those documents is nonexistent on a computer, right? I mean, there is no—there’s this very disjoint relationship between a document and the app that can open or edit it, and in fact, it’s a very fluid thing that you can change on computers, right? So, what that basically means then is that you’ve now created a system where it’s easy to have just an unmaintained wad of files that just accumulate over time in your file system and there’s no good way for the OS to reason about whether those things should be moved somewhere better or whether they should be compressed down, whether the user’s going to open it in five minutes, whether they’re going to never open it again. Does the user even know this thing is on the disk anymore? I mean, these are all difficult problems to solve and almost all of them just stem from the fact that through some quirks of history the first thing that was exposed to people on early computers was the file system because you had to be able to go in and load and execute binaries and you had to be able to pick what binary it was and where those binaries stored? They’re stored in the file system. So now the user has access to the file system, so, I mean, I think it’s, just from the get-go, it’s a very ‘computer-y’ thing and I don’t think that it’s served the computing world as well. I mean, I think that there have been as many problems caused by having a shared file system on the computer world as there have been benefits from doing it. So, you know, I really don’t—yes, iOS does things in very different ways and sometimes it can be a little frustrating, or different, or what have you, but I think a lot of those issues come up from people who are expecting it to work exactly like a computer and it doesn’t and, “Why doesn’t my phone work like a computer?” You know? Those kinds of concerns. I don’t know. Does that make sense?

Hsu: Yeah. No. That makes sense to me. I don’t know if it’ll convince other people but—<laughs>

Ganatra: Yeah.

Hsu: --it makes sense to me.

Ganatra: It probably won’t.

<laughter>

Ganatra: Won’t convince them.

Hsu: Yeah. I think—yeah. I think I can skip that question. We have a lot of questions and—okay. Say, I think you already talked about that. What was it like working with Steve Jobs?

Ganatra: <laughs> It was scary.

<laughter>

Ganatra: It was—I mean, you know, I was a nerd when I was little so he was kind of one of my boyhood heroes, so, you know, I had a chance to work with one of my boyhood heroes, or at least maybe not work with but be in a lot of meetings with, and so it was, I mean, it was a thrill, you know? It was a thrill in a good way and in a bad way.

Hsu: <laughs>

Ganatra: It was never, you know, it was—I always had a little bit less sleep the night before the days that I knew I was going to be meeting with Steve, and I was always sort of like, shields are like half up because I'm ready to get chewed out because a demo that I'm going to show isn't going to work or I'm going to have to have answers for it. You know, it got easier over time. That's for sure. I mean, the very earliest demos it was just a nightmare, you know, as far as preparing for everything that I wanted to show and having answers for, you know, if he grabs the mouse out of my hand or grabs the phone out of my hand and starts playing with it. You know, I need to have answers for why things aren't working probably the way he expects them to or what have you, you know? There's, you know, early on I felt like no amount of preparation was too much, you know? And no amount of just having a deep understanding of how this snapshot of a project works today. You know, where are the bugs that we know and things like that. But over time, it got a little easier and I think that maybe it was, maybe there was just some comfort, you know, that sort of—certainly built on my side where it was, "Okay. I don't have to have the answer to every single thing that I might be asked by Steve Jobs in this meeting, but I should really have the answers to 80 percent of the things," you know? There are going to be things that, you know, it's going to be expected like, "Well, why the hell did this just blow up when I just hit this button?" Like, you know, it better not be a surprise to me that that that thing happened too, you know? It better be something that already understood and I anticipated that he was going to hit it and, you know, and at least have an answer for it, and if I didn't have that, then at least I could, you know, it became easier for me to say, "I don't have the answer. I'll get that information," or, "I'll find that information," or things like that. Or, you know, if I didn't know something, to know, like, what was the appropriate action to take after that, and at least knowing enough to not get chewed out later on as well, so yeah. So, it was a little stressful but it was, I mean, it was a thrill as well.

Hsu: Yeah. What were those meetings like? Were these meetings where, like, everybody was, like, other executives were in the meetings too, or…

Ganatra: Yeah, yeah. It was—that was most of it. Most of it were, you know, most of the meetings that I was attending with Steve Jobs were, like, the HI review meetings, where Scott Forstall was sort of the main contact from the OS [side], meeting with Steve but then other VPs were in the room as well. Like, you know, so for example, Avie [Tevanian] was in the room for a lot of the early ones or Jon Rubinstein was in some of them or Bertrand was in some as well, and it was always—yeah. It's, you know, was a small group of execs, a Mac or a couple of different Macs, Scott sort of driving the Macs and driving the demos of what's going to be seen and things like that and Steve, sitting right there looking at it, ready to grab the mouse at any point, and sort of drive the demo, and, you know, and then you come in, set up your, you know, the demo should be all set up or pretty close to it, run through the demo, take any questions, talk through issues that you've run into or talk about whatever Steve wants to talk about, and then when it's time for someone else's demo, and I never quite figured out the rule. Sometimes I would stay in the room and other people would come in or and sometimes I would just leave and then the next person would come in and do demos, too. Those were most of the meetings. There were some—there were a couple of meetings where individual, like specific features were being discussed or other topics were discussed too. But for the most part it was the HI review meetings.

Hsu: Right.

Ganatra: Or showing progress for our demos or—One of the more memorable ones was showing progress of our hardware. And so, we had this big MX 31 development board sitting about one foot by one foot square on the table and that's connected to a modem and that's four inches by four inches square. And that's all connected up to Macintosh and that's a big 'Blue & White' or a desktop Mac with its own big screen. And we asked Steve to come in and look at it and imagine that it's all a phone, you know?

Hsu: <laughs>

Ganatra: And he did that. And so, we got through those demos like that too. So yeah, it was pretty exciting. Pretty exciting.

<01:27:27>

Hsu: So, when you mentioned you would come in and sometimes somebody else would come in and give a demo. So, like, people from your own group or you—your group doing a demo and then another group coming in and doing a different demo?

Ganatra: Yeah, like another group coming in and doing a different demo.

Hsu: Okay.

Ganatra: You know? Like the Web team coming in. Don Melton maybe coming in and doing a demo or someone from his group or I, you know, the iWork team or if there was—

Hsu: Oh, you mean completely different products.

Ganatra: Yeah, different products too.

Hsu: Not necessarily the phone, but like—

Ganatra: Right.

Hsu: —okay, so you would do a phone demo, but then another—okay, but then you would leave and then another completely unrelated product group would come in and do their own demo.

Ganatra: Yeah.

Hsu: Okay.

Ganatra: That was early on.

Hsu: Okay.

Ganatra: That was early on. Later, yeah, there were also just phone demos and—I mean. those were similar. I mean yeah, that—gosh, what was the setup of those? Yeah, it was all pretty sim—yeah; it was similar to that too. I was—I'm sorry, I was trying to remember like, as we were talking I was just thinking about like Mail and Address Book demos that we were doing. But then, and for whatever reason, I wasn't thinking about the phone ones. But, yes, it's yeah, the same is true for yeah, for all of them. It was another group is coming in or someone else—something else phone related was—for like phone demos and things like that those were sort of, when those started up, those tended to be very phone-centric. And it was, you know, there wasn't too much, you know, we didn't have a Mac person who was undisclosed on the phone coming in and looking at phone demos. It was sort of a separate world at that time.

Hsu: Right.

Ganatra: So, it did get a little bit smaller then as far as what we were demoing to Steve. And we wouldn't have people from Pro Apps, for example, coming in and demoing. But, yes, for the Mac work it was different group, different people from different groups or—

Hsu: Okay. So that was when you were on the Mac side of it.

Ganatra: Yeah.

Hsu: All right. But then later on when you're on the phone, of course, you wouldn't have somebody from the Mac side follow you, 'cause they'd be seeing the prototypes.

Ganatra: Right. Right. Exactly.

Hsu: So, it's only phone demos in that room.

Ganatra: Yeah. Yup.

Hsu: Right. 'Cause it's a locked sequestered area.

Ganatra: Exactly.

Hsu: Right.

Ganatra: Exactly. So those were smaller.

Hsu: Right.

Ganatra: Yeah.

Hsu: But then you might have, say you'd give your demo. Would another part of the phone team then give another demo of a different thing?

Ganatra: Yeah. Oh, yeah. Yeah, absolutely. Like even then if the graphics team wanted to show or the camera team wanted to show what kind of presets they were working on for images, for image capture or things like that, yeah. Those were the types of things that—you know, and obviously there's a camera on the phone and that all had to be, all of that work had to be reviewed as well. So, and that was the people who were working on the image, what attributes of the image to store, and how to have the camera set up by default to take these pictures, I mean, that was all work that my team didn't do; we just relied on the lower level camera components to handle that. And so, those were demos that we wouldn't drive. Even if the face of it was something that you saw from camera really that all the changes were happening kind of in a lower-level component and so they would demo those pieces themselves.

Hsu: Okay.

Ganatra: Yeah.

Hsu: So, you talk about the hardware being these big boards and stuff. So, at what point did things really start to shrink down and that you really saw okay, this is—this really could be a phone?

Ganatra: Yeah.

Hsu: What was that process like?

<01:31:13>

Ganatra: That was—so that really happened around, gosh, I think, for me the first time I held, like held a full self-contained phone was mid-2006. It may have been summer; it may have been midsummer of 2006.

Hsu: Wow.

Ganatra: It felt like it was not—yeah, it was relatively—there was not a whole lot of time between when that happened and when the whole world saw it. As far as the actually holding a standalone phone, the full complete piece of hardware in my hand. That happened just a few months before we announced.

Hsu: Right.

Ganatra: And that was—it was crazy. I mean, it was because, for me, this was my first time going through this whole development process on a piece of hardware that was largely being created in parallel as well, right? And so, for me, it felt like okay, we're checking in with hardware constantly, we're making sure we understand different aspects of how things are going to perform, and we understand now how much RAM we're going to have, and how long we should keep the screen lit, or like more and more of the details of the hardware were becoming clear, but it still really, to me, it felt like we were just kind of pretending or we were guessing as to whether or not this was going to work. Until I see it running on an actual piece—prototype piece of hardware, I'm going to feel like we really don't know a whole lot yet, you know? <laughs> And that was what it felt like. So even though we had this development board from Motorola that we were running on and we knew that that was an ARM-class part and we'd be roughly within that ballpark, it still never really felt real to me.

<laughter>

Ganatra: It didn't feel real to me that we were developing a phone until I saw a phone with our software on it, you know? And all that—the rest of the time it just felt like okay, what are we going to learn about that's going to mean that—what's going to be the little detail we learn about hardware—the hardware that we didn't anticipate at all that's going to have like rippling <laughs> effects through our software. It felt like what's that thing going to be? And it was only when I saw the phone running, live in hardware and was able to scroll it and see that we were getting 60 frames a second and it worked how we expected that I felt like okay, we can actually, now we're going to be able to ship this thing. We're not going to have—we're not going to have unknown unknowns, you know, to use the Rumsfeld—Rumsfeldian—

Hsu: That dates both of us with the time.

<laughter>

Hsu: So there wasn't a time where before you actually had the actual phone where you had a board that was roughly the right size or where you knew exactly this—we were going to have 128 megabytes of RAM, or whatever—

Ganatra: Well, I think—

Hsu: —or gigabytes, or whatever?

Ganatra: Yeah, I mean I think that the MX 31, I mean I think it was pretty close as far as RAM goes, and storage even. But it wasn't—but, you know, it's probably just psychological, you know? <laughs> I just wasn't used to—

Hsu: It didn't look like a phone.

Ganatra: It didn't look anything like a phone. And it looks like, it looks so much bigger and it was just boards upon boards all connected with wires and it's probably one of those things where once you go through the hardware development cycle a couple times and you realize that, okay, a dev board is going to be this big honking thing and it's relatively easy to create a much smaller package from, you know, that has all the components that this thing needs. Probably for people who have done this hardware development a couple of times, like probably the hardware team. They knew what the early iPods look like, right? And I'm sure that—actually, I remember seeing in Jeff Robbins' office like an iPod prototype and it was this big ugly thing with a giant hard drive attached to it. And, certainly, they didn't ship anything like that, right? It was this nice little handheld cool little device to use. So, I'm sure part of that is just my own lack of familiarity and lack of confidence in that whole development cycle, you know? I had to actually go through it once and see that okay, these big, you know, it is going to be possible to shrink all of this down to something handheld.

Hsu: So, it's more like one day you go one day from this big large development board to the next day you're seeing the actual prototype.

Ganatra: Right.

Hsu: It's like almost overnight.

Ganatra: For me, that's kind of what it felt like, yeah, you know? But it was such a relief to see it at the same time too. That like, okay, everybody who was speaking as if they knew what they were talking about did know what they were talking about.

<laughter>

Ganatra: We did have very capable people working <laughs> working at Apple who knew what they were doing and met their commitments. And now we have to meet our commitments too, and let's hope that we can do the same, you know? Yeah. So, I think it's just my own lack of experience there that contributed to that.

Hsu: Right. Well, speaking of which, so let's go on to talk about the demo <laughs> the keynote.

<01:36:51>

Ganatra: Uh-huh.

Hsu: <laughs> So what did you have to do to for the keynote?

Ganatra: So, if you recall, I don't remember the script itself, but there was—but a lot of the apps that Steve went through and demoed were apps that my team had developed. And so, for every single one of those every single time I saw one of our apps show up in the demo script I just got a little bit more nervous. It was sort of a—it was another point of exposure where if things went horribly wrong it would be—it would be my ass on the line because I was responsible for it.

Hsu: Yeah. Well, how early was that script worked out?

Ganatra: Early version—early work on the script, it was all late 2006.

Hsu: Like, December or was it even earlier than that?

Ganatra: From what I recall, it was earlier than that even. Like, there was already talk about it before Thanksgiving.

Hsu: Okay. November maybe?

Ganatra: I think so.

Hsu: Even October, or is that too early?

Ganatra: That might've—it may have been worked on before then, but, like, I think my first exposure to it was like mid—early to mid-November, somewhere—I think it was somewhere around there. So, yeah. It was just, it was a list of, go launch this app, go bring up this contact, go type in this, go do this thing. It was a very well understood script and it just described how somebody would go through and just demonstrate all these different features. But for every single one of those, the way it looked to me, was, okay, we need to make sure that these views are right, and we need to make sure that this Address Book is populated, and we need to make sure that, you know, all these things or all these ducks are lined up. And there were people—I mean, there were a lot of people were just following that script every single day and writing up any bugs that they ran into. Or they would follow the script, but then deliberately go off script a little bit here or there and write up the bugs they found about those too. And so—

Hsu: Because you're anticipating that Steve might do that.

<01:39:12>

Ganatra: Right. Right. And that was the thing was the script was changing over time, too. And so, we needed to—we couldn't just keep Steve on these tight rails and just sort of say. “Well, don't touch anything but these buttons.” That's just not the way it works. Nor did we want it to either, right? I mean, because the better we make all the apps work for the demo because it's all real code, it's all real live working iOS code. I mean, what you saw, what people saw in January of 2007, was real iPhone OS code up there. There was no smoke and mirrors about it. There was no, you know, there were, you know, it wasn't like “work in progress” or “Don't! That's just a still [image],” and “Don't touch anything,” and just move onto the next thing. It was all real iPhone OS running there. And so, we—and, because of that, we knew that any fixes that we get in and if we go off the script a little bit and fix those bugs too, those are bugs we were going to have to fix anyway. I mean, those are—in order to ship this thing, not just demo it; we're going to have to fix these problems. So, we may as well identify them and get them fixed earlier so that Steve doesn't run into them and we can get on to the next set of bugs to fix once those things are behind us. And so, I mean, that was another sort of dominant theme I think throughout iPhone development that I sort of picked up earlier through Mac development was any demos that we gave were real—that was all real live code running. Like we never—we didn't do any smoke and mirrors, as far as demos go, because we never wanted to give the impression that we were further along than we were. And this was work that had to be done. I mean, if you're working on smoke and mirrors that's time—you're taking time away from working on the actual product. So, if you can make it so that the thing that you're demoing is the thing that you are going to ship then all of the efforts are aligned as far as making a great demo and making a great product. Whereas as soon as you have a smoke and mirrors demo now your efforts are defused and you're going to spend some time working on the smoke and mirrors and that's going to be time taken away from making a great product. And so, we really wanted to keep those aligned as much as we could and it just made the product better conduct as we made the demo more solid, too. So, but yes, watching the demo was just this nerve-racking <laughs> nerve-racking thing. Like, every single time something went by on the demo, once Steve finished using the music player app or finished sending a text message or rearranging things in the phone app, it was just a different person in the team. Where I was sitting there were—my group was sitting around me too and it was just a different person on the team who would just kind of <sighs> —

<laughter>

Ganatra: —breathe a little sigh of relief. Or I would kind of look over like, you know, thank goodness that worked. Now the next 10 things are coming.

Hsu: <laughs>

Ganatra: Well, and actually, that's the other thing too is there was—so this demo was created and it was a script going through these various apps and doing these different things, like ordering from Starbucks or rearranging these tables and kind of showing off the UI. But then very late in the demo, and sort of in the buildup to the demo—I don't know who came up with the idea. I always assumed it was Steve himself, but there was this sort of grand finale that was done at the very end of the demo. I don't know if you remember it. I mean, I barely remember it now. <01:43:07> But for the grand finale there was—the idea was to go back through and use a bunch of the different apps together and kind of show them working together on the phone. Instead of showing here is what I do here when I'm sending an email, and here's what I'm going to do to make a phone call, and here's what it looks like when I text somebody. It's all these disjoint things. He really wanted to show how this was something that you could just use throughout your day and it's very fluid to kind of move from app to app and get things done. And so, I mean, it's a great idea and it's awesome for the demo, but if you're actually going in and creating a demo, now you've create—now you have a demo where not only do these things, all of these pieces have to work well in isolation from each other, because that's what the first part of the demo is, right, where you're going through and doing these very discrete things in these different apps? But now you have to actually make sure that the system works well enough so that you can kind of go through a bunch of the apps and do small things very fluidly as you go through the apps. So, it was sort of new demands on the—on the demo itself. Again, it was something that, as you use a phone in your normal life, this is obviously how you're going to use the phone. And so, those were, you know, any bugs that we found well, of course, they had to be fixed before we shipped. But it was just an added layer of heartburn and uncertainty and <laughs> all that on top of the demo was to have this grand finale at the very end where he then goes through and uses I forget, three or four different apps very quickly to do something at the very end. So that was one more thing, too, was even after everybody was done with their discrete demos, for some small number of people there was still the grand finale that was coming and it was unclear if that was going to—going to work well or not. So yeah, it was—it was pretty stressful.

<01:45:02> <laughter>

Ganatra: But, luckily, I mean, I think in the end I think everything worked as we expected it to. And so yeah, it turned out okay.

Hsu: Right.

<laughter>

Hsu: So, you guys were partying hard afterwards. Or possibly even during is what I heard. <laughs>

Ganatra: Yeah. Yeah. There was a little bit of somebody may have had a little thing of, a little flask of booze that was getting passed around and, you know, I'm sure I had a sip or two off of that as well as it went by. So yeah, but yeah, most of the real partying took place after that for the rest of the day, so—

<laughter>

Ganatra: —yeah, it was a lot of fun.

Hsu: So, then the demos are done but you still have to ship and the phone is still quite a ways from shipping.

Ganatra: Yes.

Hsu: So, talk about that last push and maybe some of the last-minute things that you had to deal with before you could ship.

<01:46:09>

Ganatra: So, certainly, coming out of the demo, it was understood that even though we had, you know, everybody partied and sort of had a good night and a good rest of the day after the January demo, pretty quickly after that it was understood that we still have to ship this thing. And we have never been through this process with AT&T or any other company. There is this process called, technical approval, where a phone carrier <clears throat> they receive an early build of your phone and not only do they have just a gazillion questions that you need to answer about how this phone operates and how it behaves, but then they also, the phone carrier themselves they run a lot of these tests, too. And I think it all started out of the fact that these networks that these carriers had built were these very expensive things and they had some experiences early on where people had developed phones that did horrible things to their networks, as far as using resources all at the same time, or taxing the network for questionable benefit, and things like that. So, over time, this technical approval process, it's my understanding anyway, had become more and more burdensome. And we knew that we had to receive technical approval before we could sell a phone that would then attach—would then connect onto AT&T's network. And so, technical approval, I think, we were shooting for in May, I believe was when we first wanted to send the phone in for technical approval, was beginning of May. And so, I think it was maybe the end of May, or middle of May, somewhere around then that we had finally received technical approval. I mean, up until then it was just, from January until May, it was just crunch mode the whole time. It was just a bug fix, bug fix, bug fix, get this thing ready to, you know, to be real and be out there in people's pockets and what else do we have to, you know? By then, we had these long lists of bugs that we needed to fix, software defects to fix in the iPhone code. And so, we were just—everybody was just going through and fixing things and, as you find new problems, write those up and fix them and—you know what this process is like. It can be a little—it can be a bit of a grind to get through. And, especially, some of the bugs that would come through were these bugs that could potentially impact technical approval. And because we had never been through it, it felt like this scarier thing than I think later—in later years we had sort of learned. I think you build a relationship with the carriers and they know you and you know them and so things start to become a little more smooth. But the first time going through the technical approval, it really felt like it was sort of life and death for the product and we need to be able to react to any issues that the carriers find immediately and handle those as quick as we can. So, it was definitely stressful getting through technical approval. I think we had to—gosh, I don't remember. We certainly didn’t get approval on the first build that we sent through. But I don't remember how many builds we had to do, like how many issues AT&T found before we received TA. But I know that we did at least a handful of builds of the iPhone OS on an iPhone before we received TA.

Hsu: What sorts of things were they going through? Was it mostly like just the carrier, the phone functionality? Or did it actually involve software things that you had to address?

Ganatra: I mean, it was a lot—it was both. I mean, it was both. So, there were things—like an easy example of something that could be in any part of the software—that needed to be addressed were timers. If you had a piece of software on the phone anywhere that would access the open Internet or want to make a data connection at the same time on every single phone. This was specifically a question that we received from AT&T: Is there any code on the device that will access the network at the same time on every single phone? And at the time it seemed like a funny question like, “What do you care,” you know? <laughs> “Why, maybe? I don't think so, but I, probably, maybe? But we have never looked.” And the reason was because AT&T, or Cingular before them, had run into a problem where a phone created by another company, every morning at 12:04 AM in the morning, like shortly after midnight, it would do this check. It would check for a firmware update or security update or something like that. And so, what happened was, because this phone became very popular and sold really, really well, the Cingular network every morning at shortly after midnight would just get bogged down with traffic. And it took them weeks to figure out what the heck was going on. “Why is our network going to hell shortly after midnight every single day? And then mapping it back to “Oh, well, this phone has become more and more popular and every single time we see this additional traffic, it's all from this one type of phone.” It just so happens that this phone is doing exactly that, right? It's accessing the network at 12:03, or whatever it is, checking for new software updates or whatever, same time across all different phones. And so, what it meant was every single phone was lighting up the Cingular data connect, you know, their data connection and hitting their servers at exactly the same time and bogging down their servers. And so, those are the—you know, every single rule that AT&T had was there for a reason. And every single question they had was asked for a reason. And it was usually some horrible story like that that <laughs> they wanted to make sure was never repeated again. So, it took a little while to kind of, you know, we went through that same process, too, of, well, you kind of have to think like the carriers as well a little bit and try to figure out what is—what are—how can we get the functionality we need out of the phone and still make it so that we're not taxing their networks unnecessarily as well. So, I'm sorry, I forgot what you—

Hsu: Oh. <laughs>

Ganatra: —what you asked before, but—<laughs>

Hsu: No, I think that was—I think that was about right. Yeah.

Ganatra: Okay.

Hsu: It was just I guess the kind of things that they were looking for.

Ganatra: Yeah.

Hsu: Yeah.

<01:53:35>

Ganatra: Yeah. It was those types of things or it was—it was really yeah, mostly just geared around their network and making sure that their network was not taxed. And the other thing was just customer support, making it so that it's easy for customers to kind of find a way to go and activate new features, or get in touch with AT&T, a little bit more cumbersome things, but—you know, rearrange settings to be like this instead of that and make it easy for AT&T to change the number that they call, or things like that. I guess the other thing is we created this new voicemail, this Visual Voicemail system. And so, a lot of questions were around that. But it was kind of nice because we were—I mean there—I forget what their sys—it was called ACDS, I think, their AT&T voicemail service, internally, was their name and we were one of the early customers of that or early clients of that, too. I think there were other clients of that visual voicemail system they had, but we were one of the first ones to actually expose it the way we did. And so, there was a fair bit of carrier interaction there and we sort of developed some ideas for how we want this thing to work <laughs> with other teams, too, you know? It's funny because there was, on the one hand with AT&T, we worked very—obviously we had to work very closely with them to make sure we had a great phone and that it works really well on their network, but, at the same time, there was also kind of a little bit of a not adversarial, but sort of a cautious kind of relationship as well because we knew that we wanted to sell this thing and have it work on more carriers than just AT&T. And so, anything that we did that was super tailored to AT&T was kind of viewed as it's going to be—it's going to hurt our ability to move to other carriers too or to support other carriers as well. So, AT&T, definitely, we did a lot of special things because they were the first carrier and that was just how we did things and we didn't know any different. But, over time, as we supported more and more carriers, we had to really sort of think about just sort of our ability to move the iPhone software stack from carrier to carrier and yet support them all as best as we could but, at the same time, have just one software image that's capable of reading capabilities about a particular network and reacting accordingly, as opposed to building a different image for every single carrier or something like that. That would just lead to maintenance nightmares for us. So yeah, it was interesting. But, over time, we kind of figured out where we—we didn't want AT&T support to just be pervasive through the code base as well. So, we really—we took a lot of effort to making sure that anything that was carrier specific was understood to be carrier specific and that—those are details that may change later. And, happily, we did that because as we picked up more and more carriers we could just sort of augment that layer and make it so that the rest of the iPhone stack was kind of unaware of those carrier details but could get them when they needed to, as opposed to being AT&T-ish all throughout the code base or something like that.

<01:57:12>

Hsu: Were there any, like, last-minute direction from Steve, like UI changes or app changes that needed to be made between the demo and the shipping?

Ganatra: I'm sure there were.

Hsu: Like you mentioned that the Weather widget had to be rewritten natively.

Ganatra: Yeah.

Hsu: But that wasn't your group; that was the Web group did that.

Ganatra: Right.

Hsu: But were there any things like that for your group?

Ganatra: Things like that from my group. I don't think so. I mean, I think the other big addition that happened between the demo and when we shipped was YouTube. There was a brand-new—was decided that, “Hey, we need to support YouTube and we need to have a YouTube app to support that.” And so, there was a scramble to go and create a YouTube app. But that was all—but that was not my team either. That was Richard Williamson's Web team as well.

Hsu: Oh.

Ganatra: So, I don't remember, now I’m going to have somebody from the 1.0 team disappointed that I didn't remember the huge effort that they put in on the enormous thing that they had to do between January and June. But I don't recall anything big like that other than that YouTube app. That was the biggest thing. There may have been some music player changes, but I don't—yeah, nothing comes to mind at the moment.

Hsu: Okay. So, at that point were you—I remember that the way we categorized bugs was like priority one, P1, P2, P3—

Ganatra: Mm-hm.

Hsu:—and, at some point, you just sort of stopped working on P2s and you're only working on P1s. So, were you at that point, like after the demo, only fixing P1 priority bugs? Or at what point did that cut off occur?

Ganatra: Yeah. So I don't recall that we were only fixing priority one bugs after the demo. To my recollection we were still at least fixing priority two as well. So, the broad way to think about it is a priority one bug is a bug that you would shut down the manufacturing line. If you knew about this bug, right, this is the definition we've used for the camera, if you ran into this bug, this bug—this software defect is so bad that you would shut down the iPhone manufacturing line in order to fix this bug, right? It's called a "showstopper" is another name for P1 bugs. And then P2 bugs are very important bugs that will impact the user and are really, really bad bugs. But you wouldn't—if this was the only bug that you had to fix you would not shut down the manufacturing line to fix this bug. That's sort of the broad definition of P1 and P2. And then P3 is a bad bug but maybe not everybody sees it all the time or it's—the priority tends to fall off pretty quickly after P1 and P2. So, it's my recollection that we were still working on P2 bugs—P1 and P2 bugs until, certainly, after January. And it may have been as late as a month before technical approval. So, we may have been even working on them into April, P2 bugs. And actually, I think that that's just fine because <laughs> because it was the first time we were shipping this thing. We wanted to make it so that there was a high level of quality there. And we wanted to make it so that people, even if they weren't running into the worst bug ever, they weren't running into bugs. We needed to—it was the first time that people were going to play with this thing, build an impression about how this thing works, is this something that actually fits in their life, is it something that they were going to use every day, would they recommend their friends and family go and buy it? And so, every single time you run into a bug or every single time you had a performance issue, or something like that, that sort of takes away from the experience, that's just a little less enthusiasm that a user is going to have for that thing that you spent all this time building. <laughs> And so, I think it was understood that we need to make sure that the 1.0 is as high quality as we can possibly make it. And if that means fixing P1 and P2 bugs then so be it. At least until it becomes almost irresponsible to fix too many bugs. Because at some point during software development you really need to slow down on the number of bugs you're fixing too because there's a possibility with every bug fix that you introduce a new bug. And so, it becomes this thing that you need to sort of trail off on even fixing bugs over time too because you run the risk of never hitting zero if you don't start trailing off. So, yeah. So I think we were, and I think that yeah, and that's something that I'm very proud of, is we were fixing P1—P1, P2 bugs until very late, later than you might imagine. Yeah.

<02:02:51>

Hsu: Did you have a dedicated testing or quality assurance engineers or team or were you just doing your own dog food testing?

Ganatra: Both. We were doing both. We had group of—we had dedicated QA, and we had a—and interestingly, the QA group was sort of a centralized QA. In other words, it wasn't mixed in with—on traditional—on Mac OS—on Mac OS X QA was—quality assurance engineers were ‘onesie twosie’ kind of embedded in with the engineering teams, as opposed to having a large centralized kind of QA organization that would then take on any work by a larger engineering team and having these two sort of separate organizations. Mac OS X was this merged kind of model where the QA was embedded. Well, with iOS, we went back to actually have a kind of a centralized QA organization. And so, they were working on those things. But, at the same time, I mean, as you know from OS X development, ‘dog fooding’ was a very important aspect of it as well—of product development. And so, everybody was running brand-new builds every single day and the expectation was as you are writing things you were installing and running those pieces that you were—that you were writing as well. Because if they weren't good enough for you to run then why would you ever give the poor QA group or anybody else those builds to run too, is a quick description of the logic behind it.

Hsu: <laughs>

Ganatra: Yup. So yeah, both. We did both. We do a lot of testing. <laughs>

Hsu: Okay. <laughs> So then, okay, so then the phone ships. So, what was that like that day?

Ganatra: It was crazy. We went to—I think that morning we went to an AT&T shop and we went and stood in line to go buy phones, but also to just go out and see what people were—how they were reacting to it. And how big were the lines? Were there even lines to go buy this thing and just kind of pick up on that excitement from other people who were getting the phone for the first time? So—

Hsu: You have to buy your own phones?

Ganatra: No. Well, no. No. I had—<laughs> no, we did not. Employees didn't have to buy their own phones, but, like, my wife wanted a phone on day one. So, we went and bought our respective spouses or boyfriends or girlfriends their phones. Things like that. So yeah, it was for that. And it was just fun because well, like, ever since I started working at Apple in the mid-90s, like there was always this—I was always excited when I saw an Apple product somewhere. Like you're watching a—watching a movie and somebody opens up a PowerBook and they start using it, you're like, “It's a PowerBook! Look. Check it out.” Or you're in the airport and you see somebody using a—using an iBook or something like that and it was always just like, “Check it out. Somebody who has very good taste in computers,” or whatever we would joke about. So, it was always kind of this thing that, for me anyway, and I know I'm not alone, but it's part of the thrill of engineering or part of the gratification of it is just seeing other people using what you helped to create and you kind of feed off of that excitement a little bit too. So, yeah, it was a pretty fun day. I don't think we went into the office a whole lot. I think it was me, Scott Herz, our wives, and maybe it was—I don't remember, and some other couple, we all kind of went out and we were waiting in line at an AT&T shop. And I think we went into the office to kind of trade stories about what we saw. So, yeah, it was a great day. It was a lot of fun.

Hsu: What did you think of the reaction just from the people you met on the street or in the store, but also the reviews?

Ganatra: It's funny, I don't remember a lot about the reviews. Like, I don't remember if they slammed certain aspects of it or what. But it was—I mean it was fun to see people's reactions. Ninety percent of the time it was awesome. <laughs> But every now and then you run into somebody who's—looks at it and they're like, "Well, it doesn't have an FM radio," or—

<laughter>

Ganatra: —or something like that. Or for some people you can't please everyone or—and they're very happy to tell you all the things that you left off that would make it a fantastic product. But you kind of roll with it a little bit and kind of just understand that some people are going to be like that. But, by and large, people were super excited and it was just easy to kind of feed off of. I don't know, what were the reviews early on? Do you remember?

Hsu: Well, I remember that there were—some people were talking about the lack of a physical keyboard, right?

Ganatra: Uh-huh.

Hsu: Would it be accepted? Would this be an acceptable thing that there was no physical keyboard?

Ganatra: Yup. Yeah. I remember that too. I don't remember reviews specifically saying that. I'm sure they did. But I remember, at the time, thinking that it was going to be similar to the first time that a GUI came out on a computer, you know, in the mid-'80s, where it was this thing that made the GUI, you know, the Graphical User Interface, it made the computer more approachable, and it made it a little bit more user-friendly if it was designed properly. But, at the same time, when it first came out, when the Macintosh first came out in the mid-'80s, the big slam against it was, "Yeah, these GUIs, they're nice and all, but it's not really well-suited to the serious work of a business computer." You know, there was all this—that was sort of the criticism of the GUI at the time. And I remember when the iPhone was getting slammed for not having a physical keyboard, I remember it feeling an awful lot like that. Or people would slam the mouse, you know, they would say that, "Well, because you have to take your hand off of the mouse all the time, it's going to impact the productivity of a serious worker, and so, therefore, serious workers shouldn't have mice on their computers." You know, was sort of the logic that people followed. And so, I remember a lot of that same, you know, a lot of echoes of that sort of coming back with the iPhone, where it was, "Okay, well, yeah, this is cute, and it makes it so you could whatever have a separate keyboard for different things if you wanted to, but you know, it's really not going to be, you know, for a power business user who needs to send messages very quickly, they really need a physical keyboard, and are they really going to be well, you know, is this really well-suited for them?" It felt very similar to that. Like, “Yeah, this is a cute toy and all, but it's not a serious work machine," you know? And I think it was just a matter of using the thing. And, of course, we had to add additional functionality, too, to kind of pick up some enterprise clients, and we did that later thankfully. But, yeah, early on, I remember that, and it kind of felt like, "Well, yeah, we probably do need to make the keyboard better in some ways, but at the same time, this is—hopefully this is something that people will get used to." And I knew that just from my own use, you know, we were using it every day, all the time anyway, just as part of the dog fooding process. So, it felt like a good fit for me to get my own work done up until then. But, you know, you never quite trust whether you're a good gauge for other people anyway. But it felt good to me and so I was confident that people would come around, but yeah, you're right. It wasn't a slam-dunk, and there was definitely a lot of that. That was the buzz at the time was that, "This isn't a serious business phone. You know, it might be fine for rich people, or what—," I forget what Microsoft said at the time about this being a very expensive device or what have you. But you know, "For the serious business of—you know, for the serious task that you need to run a business, why would anybody ever have an iPhone?" I don't know. You kind of—for me, I was used to those arguments. And so, it just felt like, "Well, I'm going to make it work as well as possible, and hope people come around." So. Yep.

Hsu: So then, next, what did you have to work on immediately after?

Ganatra: Immediately after, the next thing we had to work on was the European rollout. I think there was already—there was more and more buzz that was coming about the fact that there was not a third-party app story.

Hsu: Right, okay. So, okay, that gets into the maybe the other question that I really want to ask was that initial—the initial story was web apps only for third party developers, right?

Ganatra: Yep.

Hsu: And that was the official story. Okay, so yeah, so we're talking about the SDK and the App Store. So, from your perspective, at what point was the decision made to actually do that, right? It seems like you were mentioning that people outside the company were already talking about the possibility of Apple opening it up. The official story was that this was not going to happen. Was that different internally? Or from your perspective.

Ganatra: Let's see. So initially, it was not different. And there were definitely a lot of requests that came between January and even when we shipped. You know, people coming and saying, "Hey, you know, web technologies aren't going to—they're not going to work as well as what you demoed, so why is that the answer for third parties?" And, you know, I think more and more over time, those questions must have just worn away at Steve and other people. I wasn't in the room when he made the decision to actually go and support native app development, but I think that more and more at Apple, there was an understanding—I mean, there's always been this understanding if you've created two technologies, one for your own use, and one for third parties, or for somebody else to use, then that must have been for a reason. It must be that either the things that you're doing are so different from what you expect third parties to do that you need to have some brand new, you know, some technology that's different, or you think that what a third party will be doing or could do is not as important or as compelling or interesting as what you may be doing on your own. So that's why you have two. In either case, it's not a great answer. You're not in a great position if you have kind of a different solution for yourself than you have for your third-party partners. And so, I think more and more, over time, that became the kind of—that came to kind of wear away at internal management as well. Which is sort of the understanding that, "Well, if we're creating a brand new—," you know, we had just created, or Richard's team had just created this YouTube app, and you know, just by default, they created it as a native iOS app. Why? Because if they went and used web technologies, it would have been a much different user experience. It wouldn't have been consistent with everything else on the phone. Scrolling would have been different. There would have been a lot of functionality that they would have to implement by hand, using the web technology solution. I mean, just lots of different reasons why it just was not—it would not be a great choice for Apple, even if we were the ones who provided that third party solution, it still wouldn't be a great choice for us to use that solution either, and that we should use the thing that we've internally developed and demonstrated that I works pretty well and we have a lot of confidence in. So, I think that over time, again, those were the things that started to wear away and early on there was—it's interesting, though, because the first justification—the earliest justifications for creating third-party apps were to satisfy professionals. You know, it was really one of the big apps that we were targeting, kind of internally, and I don't know that we'd ever reached out to them. I think we actually did. I think Apple actually did. Was this app called Epocrates for Palm devices. So Palms were these personal digital assistants that then—that they tacked a phone onto, and they had a third-party SDK, they had a native app develop—you know, a native app story, and this company went and created this database of medical information for doctors to be able to use out in the field to be able to pull out their PDA and look up different details or look up either medical information or look up information about a disease, or you know, about a different type of medicine or how it might mix with other medications or things like that. And so, really our very first—you know, our first target was Epocrates. And we were kind of thinking, "Well, what if there was a doctor and they're walking around with a Palm—," I forget what they're called—Treos I think it was, you know, which is this Palm Pilot with a phone attached to it. You know, "What if a doctor wanted to replace their Treo and they were running Epocrates? Well, what would they do?" And so, there was some internal work around just kind of going through the exercise. Well, what would it mean for Epocrates to actually ship a web app? And I think it became—very quickly we realized that either the—in order for that company to satisfy—to create something like Epocrates for the iPhone, they would have to do an enormous amount of work, right? They would either first have to figure out how to truck all of that data over to a phone, and make it so that this web thing can present it instantly. Or they would have—we would have to have some persistent connection to the web, and be able to pull up that information from the web app. But that might be different from how the Treo-based Epocrates works. So, then that would just be a huge amount of work for them to do on their end. So, I mean, ultimately it came back that if we wanted to use the technology that they've provided—oh, and by the way, once they had done all of that, then they were going to have—then what they would end up with, was an app that behaved nothing like any other app on the system. It would probably take longer to launch. It probably wouldn't have the nice smooth scrolling. It wouldn't have all the features and functionality that other apps on the system have. How would you go in and control settings? You know, all these little de—you know, there was just these little questions about issues around creating this web app. And so, I think that was another case where just kind of going through the mental exercise of, "Well, what if we did want to go and target this client, and wanted them to make a really great web app, what would they have to do?" Answer: "An enormous amount of work, and they'd end up with something not great anyway." So, I mean, you know, I think we went through enough examples like that until we realized that really the right answer here is to release an SDK. And luckily that was, you know, and Scott Forstall, to his credit, he knew what we were—you know, what these projects looked like and how we were already building the software. He had a good understanding that, internally, even though we didn't have a third-party SDK, internally, we were developing these things as though we had—you know, using our own internal SDK. So then the amount of work that we would have to do—we already have an SDK, basically. So, we would have to do some sanitizing and cleaning up and getting some interfaces ready to share with the outside world. And Apple takes those interfaces very seriously. So that's an enormous amount of work. You don't just open things up and let people do what they want, and then now you have a huge problem later on. Apple understood, there's a lot of work you have to do upfront to make sure those interfaces are—you're going to be happy with those interfaces for ten or fifteen years or twenty years. So, there was a lot of work to do, but it was the right thing to do was actually ship an SDK. And I think that, you know, history has proven that, but, early on, I don't think it's so crazy to think that this thing was going to ship and not have an SDK either. If you look at the iPod, which was this, you know, the first example of an embedded product that was wildly successful for Apple that didn't have an SDK. You know, now, of course, it's a very different kind of device, but, later on, there was this kind of a funky SDK that did ship even for iPods that allowed games to be developed for it. But, early on, it was still a success even without the SDK. And so, I think that that's how people were thinking about the first iPhone. They were thinking about it more as an iPod that could make phone calls, rather than, you know, this whole computing device that was a platform all on its own.

Hsu: Mm hm. So how quickly was that decision made?

<02:21:44>

Ganatra: It was made, I think, I mean, it's hard for me, because I don't remember the first time that anybody actually said, "Okay, we're doing it, and let's go." The first time I heard about it was after the 1.0 release was that, "Okay, we're doing this," you know? But at the same time there was already work that was taking place to support European carriers and, you know, there was additional international support had to go in for that. And a little bit of building the layers in to kind of separate the iPhone OS from the carrier—you know, from AT&T and make it just kind of a more general thing that we could support, so we could go and pick up Deutsch Telecom Support, and Orange, and O2, and Vodaphone eventually, and those companies. So, we had plenty of work to do <laughs> outside of the SDK. But the SDK work did also pick up. From what I recall, it was after 1.1 shipped. So, I think it was like late 2007, that was when we had started in earnest on the SDK work.

Hsu: Okay. So, you mentioned the enormous amount of work to sanitize the APIs, the UIKit, APIs for public use. Could you just describe that process? And one thing I wanted to sort of—I think I heard that internally you—the team had a UITable class. But then externally, there was a UITableView class, and there was a period where the internal apps were still using the internal thing, and not the external thing.

Ganatra: Yes!

Hsu: So, maybe you can talk about that a little bit, yeah, as well?

Ganatra: Yeah, so even though I went through and described how, <laughs> you know, Apple hates it when we have technology that is being used internally, and that it's different from what's being used externally, we actually did, you know, even though we came around to the, you know, to creating this—to having the desire to create this native SDK, and making it so that that was the way that third parties would implement their software was using something similar to what we were doing, so, therefore, our interests were aligned, it just so happened that for the first version of the SDK, and it may have been the first couple of versions, even though we were all Apple internal apps, built-in apps and third-parties, even though they were all using a native SDK, it just so happened that the classes that a lot of the Apple internal apps were using, were different. The implementations of a lot of the objects, of a lot of the UI widgets were different between what Apple was using privately, and what was available in the public SDK. And you brought up definitely a sort of a standout example of that, which was UITable was the class name of the private classes that we used internally to implement things like lists within the UI. Like scrolling lists where you can pick an item. And where the external—or the third-party class name was UITableView. So, the UITableView was the one that was nicely sanitized, nicely cleaned, it had method names that we wanted to support for fifteen—ten/fifteen years, what have you. And that was what our expectation was that that was what third parties were going to use. Actually, our expectation, our long-term expectation, even then, was that internal and external code was all going to use UITableView, but there was this period of a couple of SDK releases where we were using the unsanitized, private only internal versions of UITable, that just so happened to have far better performance characteristics <laughs>, because we were taking a lot of shortcuts in the implementation of UITable that would not be appropriate to take in a sanitized, cleaned up TableView. And so, there were, you know, early on there was definitely some people on the outside who realized that, "Hey! This is funny! When I make an app that looks an awful lot like the Address Book app, or if I make an app that looks an awful lot like the music player app, where the cells within this TableView are very similar—you know, are similar to what I would see in one of Apple's built-in apps, it looks like I'm getting, you know, my scroll rate is significantly slower than what the Apple internal scroll rate is." And some clever engineers then went in and figured out that, "Aha! Apple's using these private classes that are fast and must be awesome and have all kinds of great stuff in them, and they're sticking us with these crappy external classes to use that are slow and porky and what have you." And so, it did—it took a couple of releases before we had transitioned over to those public classes, too, because again, when you live in a dual world like that, that there's also not the—you know, you no longer have your interests aligned with the interests of third-parties when you have this dual world. And so, if you see a performance slow-down in the public version of these classes, well, it's not going to hurt you at all, if you're using the private ones. And so, yeah, that is an important bug to fix, but it's not going to be in your face every single day. And so, if it's not in your face, and if it's not bugging you every single day, it's naturally going to drop in priority. Once it drops in priority, now you've got this dual world for much longer than you wanted to have it to begin with. So, in my mind, it was very important for us to, you know, once we had this public SDK, for us to start using, being a client of the public SDK as well, just so that we can start addressing, you know, a) we can get rid of duplicate code, right, because now we had this private classes and we had the public classes. But, more important, our interests are now aligned, and now, just like third parties have been squawking for a year that TableView scrolling is slower in third party apps than in ours, now, all of a sudden, we've made that—that was the problem that third-parties have, we've now made that our problem, too. And once it's Apple's problem internally, it's something that we have to address. We can't go backwards in performance and have it have scrolling be slower, or be more porky in the new release than it was in old. And so, once we were all aligned that way, then we could do all of the extra work it needed to make TableView perform very quickly and as efficiently as UITable or almost as efficiently as UITable, therefore, everybody wins! Now third parties get to speed up, and Apple apps are no different visually than they were to anyone else.

<02:28:58>

Hsu: But why did that happen in the first place? Was it just lack of time that—

Ganatra: Oh, that UITable was created?

Hsu: Well, no, but that there were these two parallel. Was it just the—there wasn't enough time to work on the public SDK, and move the internal apps to the new SDK at the same time?

Ganatra: Yeah, yeah, I mean, it's time and just managing the resources, you know? Because as soon as—I mean, we understood that as soon as we switched our apps over to the public SDK calls, we're going to immediately take a performance hit. We knew that! <laughs> We knew that that was going to happen. And so, and I think that there may have been—I need to—well, I don't know how I would check this now. But I think that there were one or two clients of the public SDK, even though it was understood that they were going to be slower and not work as well as the private versions, and, even in the very first version that we shipped, I think there were some apps that used those public APIs. But it just wasn't the mainstream apps. Like it wasn't Mail, and it wasn't Messaging, you know, Messages. And you know, kind of the core built-in apps. There just wasn't enough time to both release a public SDK and do all the work that we needed to do to support internationalization and new carrier support, and cut over to these new APIs all at the same time. I mean, these are thing—you know, we only had a certain number of engineers. We knew that we had to be aggressive on cutting off how much functionality we add to any individual release because we wanted to keep the quality level very high as well. And so, we had to make some short—we had to take some shortcuts like that, kind of along the way as we went to, you know, as we developed this thing. You know, some little ugly shortcuts kind of, you know, that we could snapshot and ship, something that to the customer seems like a very high-quality release, even though internally we know we've got some maintenance work we need to do. But I think that's just the reality of software development, too. There's always going to be maintenance work to do. There's always going to be more work to do than time that you have to do it. And how can you be intelligent about what you do for this release to make it a great release, but not tie your hands in the future, or make it so much more difficult to get your work done down the road, too. So, yeah, it's just trade-offs. Trade-offs, yeah, we just didn't have time to do it all, all at once.

Hsu: So, then going back, the actual work to create the SDK, like what did that involve?

<02:31:51>

Ganatra: So, a lot of that work was Toby, Toby Paterson's group. So that was another team entirely.

Hsu: Oh, really?

Ganatra: Yeah, yeah, that was Andrew, Andrew Platzer did a lot of that work as well. And they weren't--

Hsu: So, they weren't in your team at all?

Ganatra: Toby was not on my team. Andrew was not. I think Toby and Andrew reported directly to Henri, so they were peers as well. Even though we had developed the first version of UIKit, and Scott had worked a lot on it, and we had all been contributing different components to it, I think Andrew was "the" owner of it, of UIKit, and so once—

Hsu: Oh, as an entity.

Ganatra: As an entity itself, yes. And so, once the work of the SDK came along, that was now a lot of work that had to happen just in UIKit on its own, right? I mean, I guess there were other APIs that we did publicize, too, like Address Book UI, and Messaging and things like that.

Hsu: But Andrew was on the team for 1.0, too, right?

Ganatra: And he was on the team for 1.0, yes. From what I recall, he reported directly to Henri, though.

Hsu: Huh. Interesting.

Ganatra: Yeah.

Hsu: Even though he worked on the same stuff that your team worked on.

Ganatra: Yes, yes.

Hsu: Huh.

Ganatra: We, yeah, right. I mean, it's funny, I guess, in hindsight, it is kind of funny. But, I mean, we were all—everybody was sitting so close to each other anyway. You know, I was in Andrew's office for countless hours anyway. You know, it was sort of—there was the organization, and then there was how we did things as well, you know?

Hsu: So it didn't really matter who reported to who.

Ganatra: It kind of felt that way a little bit. You know, it was—I mean, we weren't—the teams were all small enough, and everybody knew everyone else, and, so, there wasn't this sort of the what you might see in more traditional companies, where you escalate something up to a common manager, and then things are discussed there, and then notes bubble down, or what have you. It was Apple after the NeXT acquisition. It didn't really work that way anymore. It was very much, "Go talk to the engineer you need to go talk to," you know? There's, you know, yes, something needs to be bubbled up to their management as well, so they know what's going on, if management is all on the same page as far as, "Ship this thing as fast as you can, and this is your top priority," then management is largely in the loop anyway. And so, yeah, you don't have to worry too much about kind of the organizational minutia.

Hsu: Right, right. But then, okay, so then Andrew Platzer and Toby Peterson?

Ganatra: Paterson.

Hsu: Paterson, Paterson.

Ganatra: I think, yeah, P-A-T-E-R-S-O-N.

<02:34:46>

Hsu: Okay. So, they're essentially in charge of publicizing the UIKit, turning it into an SDK.

Ganatra: Yeah, yup! Yeah, exactly. So, you know, and coming up with UIColor, and you know, all the method names, and all the things that are going to be, you know, creating the classes in such a way that they're maintainable later, and they work really well, today. I mean, Andrew is—anybody with an AppKit background in that I trust way more than I trust an app developer to come up with classes that are going to be reusable and can sort of stand the test of time. And so, I think having Andrew be largely in charge of that was probably a smart thing for Henri to do.

Hsu: Right.

Ganatra: Yeah.

Hsu: So then what was your team's role in supporting that effort?

Ganatra: So we were, I mean, we definitely, so we did have our own APIs that we exported as well. For things that were more app-centric, right? So, like for Address Book and for, you know, there's a People Picker. Like a Contacts Picker type UI that we had available. There's in Mail, you know, the ability to send—to compose and send messages. There's public API for that. So, we were busy with those things. In addition to doing API review with the UIKit with Toby and Andrew, and just, yeah, reviewing new class names, method names, and you know, where we could we would cut over to things as quick as we could just to get more airtime on those pieces. But we certainly didn't convert to everything in the first—when the SDK was first available.

Hsu: Hm. And how much like in terms of the actual mechanism of the App Store, did—was that all a separate part of the organization? Or—

<02:36:55>

Ganatra: Yeah, so there was the backend, and there's sort of the—I mean, there’s sort of the server end of the App Store. The server-side development work. And then there was kind of client side, right? So that was client side development work for App Store and the Music Store were done by my team by an engineer who was previously working on like the music player. But, yes, but because such a large component of that work is server-side, the server side team was heavily involved, too. And so, we were in regular meetings with them discussing how these things should work, and, I mean, it was just a cross-functional effort, you know, all around, from, just the act of browsing a store, and then being able to download something, and authenticating for download, and entering payment information, and now you've downloaded some bits, and how do you open them up? And, you know, make it so that you can execute those bits? And do you have the right to execute them? Has that app been revoked? You know, that there's and then how do you go about installing it? You know, there's a lot of work that goes in, kind of at all different levels. And the App Store itself is kind of sitting right in the middle, so, yep.

Hsu: Right. So, then those decisions about, you know, you have to have a provisioning profile, and there's a 30 percent cut to Apple. All those things are like—those decisions are all made somewhere else.

Ganatra: Correct, correct. Yep, I mean, those were all on the business-side, and they were all sort of parameterized, so you know, well, I mean, the 30 percent, we never saw that anyway, right? That was all entirely on the other end of compensating developers and things like that. So, yeah, that was really just more on the business-side of the iTunes Store, and there's a whole other organization managing that. Like Eddy Cue's org was managing that side of the effort.

Hsu: Right, yeah, yeah, yeah. So, when you mentioned the server team, was that like the same team that had done the iTunes Music Store?

Ganatra: Yep, yeah. It's Eddy Cue's org had worked on both of those.

Hsu: Okay.

Ganatra: Yep. Yeah, and so behind the scenes, they worked very similarly, I mean, they were just assets that you could browse and download, and they ended up in slightly different places in the UI and in the file system, but largely an app is similar to an album, as far as downloads and things like that goes.

Hsu: I think like a lot of the server stuff was done in WebObjects, correct? Or was that—had that changed by that point in time?

Ganatra: I believe it was still in WebObjects. I'm not 100 percent sure, though. But I believe it was, yeah. Yeah, there was definitely a lot. A lot of the early store, you know the Mac Store was all done in WebObjects, too, yeah.

Hsu: Yeah.

Ganatra: Yeah, it's pretty cool. I never learned a lot about WebObjects, enough to understand it, you know, beyond just very superficial.

<02:40:10>

Hsu: Mm hm. Let's see. So then after the SDK is public, how much—how big was that change overall for just the platform and for your team to port to that?

Ganatra: I mean, for the platform, I mean, it was the creation of the platform. <laughs>

Hsu: Well, right.

Ganatra: <laughs> That's pretty big, but yeah. Sorry, what were you—

Hsu: But for your team, the nature of your work change at all now that you are supporting third-parties.

Ganatra: Oh, yeah! Oh, absolutely. Absolutely, yeah. I mean, we now, you know, where previously we answered to customers, sort of end user type customers about issues with the app and the UI or functionality, or things like that. Now, we're also dealing with developers. Which can be less enjoyable at times, too. <laughter> Well, it's a whole different bag, you know? <laughs> I think, like I had mentioned before, engineers are not shy about letting you know what they want, or what's causing them grief. And you know, developers at other companies are no different. So, yeah, it was, you know, but that was just—you know, now we had an API that we had to manage as well, and maintain that, too. And so, we try—you know, we had feedback from the UI team and a lot of people on the API that we—APIs that we had. But yeah, ultimately, yeah, the job did change, too, that we had to deal with developers as well. But yeah, I mean, it's the right thing for the platform.

Hsu: Yeah.

Ganatra: Yeah, so it was fine.

Hsu: So then for iOS 3.0, what's the big change? What's the big push there?

Ganatra: Oh, boy! This is a quiz, isn't it? <laughter> Let me see. Okay. I think MMS. I think a lot of the enterprise work.

Hsu: Oh, yeah, supporting Exchange Servers?

Ganatra: Yeah, yeah. Exchange Servers. I think blocks may have gone in 3.0.

Hsu: Oh, really?!

Ganatra: I think.

Hsu: Objective-C blocks?

Ganatra: Yeah, Objective-C blocks. Maybe ARC. Might be. I'm not sure. I don't remember now. That might have been later, actually.

Hsu: That may have been 4.0 or later.

Ganatra: Yeah, that was probably later. But yeah, Enterprise and MMS, I think those are the two big ones that stand out in my mind.

Hsu: Yeah.

<02:42:52>

Ganatra: And then, internally, there was iPad work as well.

Hsu: Right, okay, so iPad. So, let's talk about iPad.

Ganatra: Uh huh!

Hsu: Okay, this is obviously a huge change. You're supporting a completely new device now, with a completely different set of screen dimensions.

Ganatra: Yep.

Hsu: And somewhat different user interface interactions now. So, what work did your team have to do to support the iPad?

Ganatra: Oh, I mean, yeah, we had to do—we had a lot of work. Early on, the work that we had to do was convincing Steve Jobs that it was going to be a lot of work. So, in other words, Steve was convinced that what you could do was just take the iPhone user interface and just expand it out to be the size of a 9-ish inch screen and just ship that, you know? "And so. what's the problem there?" And a couple—you know, and there were a couple of meetings where one or two people would describe some specific problems with having lists that scroll on the screen that are the width of, that are six inches wide, and you know, you just have a name that's taking up two inches on the left. And what is that going to look like? But it was really, I think, it was the HI team that really drove it home with just an example: Images of, "You know, well, you're saying that this is what you want. But, really, you're probably—this probably isn't what you want," you know? "You're saying that this is all the work that we need to do, but if you look at this, it's not going to create an experience that you're—it's an experience you're not going to be happy with." And you know, when you look at like the big TableViews and it becomes very clear very quickly when you're looking at a UI like that, that you're wasting screen space. That it's really a big waste of screen. Kind of getting back to what you were talking about earlier that when you've got this big honking list, and you're only taking up the first quarter of it, of any cell in a scrolling list. Well, I could be doing something better with those three-quarters of that empty cell. Like why is it just blown up like this? Why doesn't it show me, you know, if we're talking about Address Book, and it's a list of names that are just taking up the left two to three inches, why doesn't it show me the groups right next to that, too? Or when I tap on one, don't scroll me all the way over to a full screen view of that card. Maybe I just want to see a representation of that card off on the side instead, so I'm not scrolling around so much, you know? So, there was a lot of learning that had to go in, too. And I think, by the way, the other thing was if you have the iPhone UI just blown up like that, you're now—the iPhone is just doing screenfuls of transitions, and every single time the iPhone does a transition, if it's doing a screenful, it's quite different—you know, the number of pixels that you're moving on a three and a half inch screen is a lot different than the number of pixels you're moving on an eight or nine inch screen. And so, if you're doing these full-screen transitions, it's going to be a lot more disruptive if you're looking at them on an iPad, than if you were just dealing with it on a phone. And so, a lot of these things just had to be demonstrated to Steve to kind of show him that, "Eh, you know, these things are not gonna—there's what you're saying, but really let's just be honest, this is not going to create a great experience, so let's just get on to—instead of just blowing up the current iPhone UI, and pretending that this might be something you might like, let's just get onto the business of actually designing some of these apps and putting in some things that are a lot closer to what you're probably going to want to see in the—you know, on an iPad instead of just a blown-up iPhone UI." So that was the early work that happened. A lot of the early work. Then there was the matter of once we had these designs, actually working with the OS X team to actually implement a lot of that, a lot of that work, too.

Hsu: The OS X team.

Ganatra: Yes. So, the OS X.

Hsu: Oh!

Ganatra: Specifically, Don Melton's group. And so, we—

Hsu: On Safari?

Ganatra: And so, the Safari team, well, by then Don Melton's team had grown to not just Safari, but Address Book, Mail, Chat—

Hsu: Oh!

Ganatra: And I think like some low-level—it might have been like the Carbon team, as well. But I'm not sure. That may have still been—yeah, I'm not—I don't remember if that was under Don. Anyway, but it was more than just Safari. But the thought was that we could have them come and help implement some of these new apps instead. And so, Calendaring was this brand-new app developed by the Calendar team to run on the iPad itself. Same thing with Address Book, that was a whole brand-new app, that was managed and developed by Don Melton's team. And we actually, depending on the app that you're—or depending on the platform that you're running on, we would either launch the iPad version on iPads, or phone version on phone.

Hsu: Oh, okay!

Ganatra: You know, and that was kind of how it broke down.

Hsu: So, okay, so it's like you have the team—so it's like the Address Book team in OS X is also doing the—well, that's not quite the case. Because you already had Contacts on the phone, but like Calendar.

Ganatra: Right.

Hsu: The Calendar team on OS X was now also developing the iPad Calendar.

Ganatra: Right, right. Even though there was still a phone Calendar app, you know, the iPad Calendar app was this whole—its whole own thing that was developed by Don Melton's team.

Hsu: Oh, okay. Huh.

Ganatra: So, yeah, so a lot of those early versions, and it may still be true today. I'm not sure, but, yeah, a lot of the early versions of those iPad apps were completely different apps that were developed solely for the iPad. And then—but depending on what system you were on, the Springboard would launch the correct one.

Hsu: Huh.

Ganatra: So, yeah, so that's how that worked early on. That's how we shared was we just kind of handed over entire apps to Melton's team to let them develop the iPad versions.

Hsu: Right, okay.

Ganatra: So, yep.

Hsu: And then it was like, so the—and it's the HI team came up with some of the iPad specific UI elements, like the split views, and the popover thing.

Ganatra: Yep.

Hsu: So that you wouldn't be sort of pushing down and transitioning these screens a million times. But you would just—a portion of the screen, something would pop up, like a little mini window.

Ganatra: Right, exactly, exactly. Yeah, and so yeah, right. Those are the types of things that had to be developed over time where—you know, when you have a nice big screen, obviously you do things different than if you have a very, very small screen and there's only so much you can show. And so, it was nice to be able to take advantage of the fact that, "Okay, if you were just going to change this one field, you know, don't make the poor user wipe their whole screen to kind of change that one view, and then wipe the whole screen to go back. Just, you know, bring up something that, you know, very small, and not very intrusive that allows the user to do that instead."

Hsu: Right.

Ganatra: Yep.

Hsu: And so did your team implement those things?

Ganatra: Let's see? Popovers, I think that was—no, I think that that was Andrew—I think that was the UIKit team.

Hsu: Okay.

Ganatra: That did that. I know that Evan was involved. Evan Doll was involved early on in sort of the master-detail view kind of relationship. But I don't remember when—you know, if we had handed that off, or where that went. You know, he may have developed the whole thing, but just for the UIKit. You know, like I said, you know, it's funny, because there are different components—like Alex Aybes and Evan, I think, were both working on ViewController support within UIKit. But then other things got handed off to the UIKit team. It was sort of a mishmash of who worked on what. So, I think the same thing happened with—that's why I'm not completely 100 percent clear on who worked on what for iPad in UIKit. I know that Evan did some of it, even though Evan was on my team, not on Toby's.

Hsu: Right, oh, okay. So, by this point there's—so the UIKit team is now a full separate team—

Ganatra: Right.

Hsu: —and your team is the Applications team.

Ganatra: Correct.

Hsu: And so your—now the relationship is less like when originally on iPhone you were developing the Apps and the Kit, it's simultaneously. But now there's a clear separation, where there's a Kit team and there's an Apps team, and you're more clients of the Apps team—or of the Kit team, of the UIKit, in a way.

Ganatra: Yeah, exactly, yes. That did morph over time. It changed from, yeah, we did—there was no UIKit, there was none of these shared facilities or anything. And so, we had to invent a lot of those. But then once we actually had enough work on that side to do, it was—actually before that, you know, Andrew came along and was already working on ScrollView. Like working on scrolling, and making those different designs—you know, the UI widgets work as well as they do. So yeah, so already Andrew was there, and working on parts of UIKit, but over time, then, yeah, Toby came on. Toby started hiring more people, and then the UIKit team sort of filled out more as you described.

Hsu: Oh, okay.

Ganatra: Yep.

Hsu: So initially it was just Andrew by himself.

Ganatra: Right.

Hsu: And then Toby came on sort of as a manager to manage?

Ganatra: Yes, that's what I recall, yeah.

Hsu: Okay, and then it grew out from there.

Ganatra: Right, exactly. Exactly, yeah. So, Andrew never reported to me. He just reported straight to Henri, but then I think Henri brought in Toby, and had Andrew report to him, and then Toby expanded the team from there.

Hsu: Okay.

Ganatra: Yeah.

Hsu: Okay, <laughs> yeah, that makes a lot more sense now.

Ganatra: Yeah, I know. <laughter> Sorry, it's, yeah, all these things were moving and changing and morphing over the years, too.. <laughter>

END OF THE INTERVIEW