Oral History of James Gosling, part 2 of 2

Interviewed by:
Hansen Hsu
Marc Weber

Recorded April 22, 2019
Mountain View, CA

CHM Reference number: X8971.2019

© 2019 Computer History Museum

Hansen Hsu: It is April 22, 2019 and we're back with James Gosling.

James Gosling: Hello.

Hsu: Hi. We were sort of talking about or finishing up the story of your time in graduate school. So can you talk about your dissertation committee?

Gosling: Yes. So I had an odd arrangement in that I had two thesis advisors Raj Reddy and Bob Sproull. And then on my thesis committee was also Ivan Sutherland and Guy Steele.

Marc Weber: No pressure.

Gosling: Yeah. Ivan was a lot of fun.

Hsu: Could you talk about your dissertation topic?

Gosling: The title of the thesis was “The Algebraic Manipulation of Constraints”. And I always liked building-- liked editing drawings and it always seemed odd to me that most drawing editors were just like lines. And at some point I had seen one of the Sketchpad videos or something. The fact that Ivan had put constraints into it and I thought, well that was kind of cool. And I started looking at systems that tried to be kind of like that. And in Ivan's thesis he uses kind of two techniques, one that he called just constraint propagation, you know, you wiggle this and then that wiggles and that wiggles and that wiggles. But that sort of falls apart when there are cycles. So he would go from that to numeric relaxation which is, you know, computationally really, really heavy. And, often, would lead to surprising results, usually not surprising in a good way. So I decided well, could I do a constraint satisfaction system that was based on symbolic algebra rather than doing it numerically? And so that's what I did. I wrote lots of stuff to do symbolic manipulation and trying to take the graph where there are cycles. And then rather than trying to solve it numerically, turn the subcycles into expressions because the subcycle ends up being a system of equations and then try to solve the system and turn it into one equation. And then try to come up with-- I tried various schemes for what people would consider surprising versus not surprising because all too often these constraint systems are under constrained so you kind of want to change as little as possible. I kept finding that most of the systems of equations had, you know, obvious solutions at like 0, 0, 0. And, you know, that was never very satisfying. You just find your whole diagram just kind of go <blows raspberry>. And that wasn’t very good. So it's like trying to find the minimal solution that satisfies the constraints or the minimal change and so that was kind of fun.

Hsu: And with Sproull and Sutherland on your committee it tilted you heavily in a graphics-oriented direction?

Gosling: Yeah. Yeah, well neither of them-- Ivan is very famous for his work in graphics, but-- well, both of them really are all over the map. They're pretty amazing guys and they do all kinds of things. And, you know, my whole thesis committee was a real privilege.

 © 2019 Computer History Museum

Hsu: And the other thing is that even though you did have Ivan and Bob who had implemented systems, your dissertation still had to be theoretical.

Gosling: Yeah, that was kind of-- well, I built a system. I mean I had to build the code, too, but-- I had tried to get several thesis proposals kind of like through the system at Carnegie. And one of the pushbacks I got was always it's just not theoretical enough. So there had to be something that felt sort of theoretical to people. So the underpinnings of what's now the Java virtual machine I had been hoping to get that-- because I thought that that was a really interesting set of ideas and maybe that would make a good thesis proposal and it's like-- I don't know. Raj and Bob were like ehhhh. You know, the faculty has to sign off on the topic and that one just had not-- it didn't have enough opportunities for the equations and theorem proving and crap like that.

Hsu: We touched on that little bit last time, the virtual machine stuff so that was for this emulator that you did for the PERQ to run PERQ software on the VAX. How familiar had you been with things like UCSD Pascal and the Smalltalk byte code precursors?

Gosling: Not a lot. Most of what I learned about UCSD Pascal I had learned from picking apart PERQ binaries because the PERQ folks had just used the UCSD Pascal p-code and there were things about it that were great and things about it that were not so great. Although, I was looking at them through a very strange lens namely I wanted to turn it all into VAX machine code and there were places where I had to work extra hard and places where it worked really, really well. And then I talked with some of the Smalltalk folks in the following years about the Smalltalk byte code and, you know, I really hadn't used Smalltalk much at all. I had played around with it on a Xerox Alto a little bit.

Hsu: Which Smalltalk people had you been talking with?

Gosling: Peter Deutsch mostly because he was at Sun at the time.

Hsu: OK. So you finished your PhD in 1982 or ’83?

Gosling: Eighty-three.

Hsu: We talked a little bit about this last time. You then went on to IBM.

Gosling: Yeah.

Hsu: So then at the time at IBM you were working on the Andrew Window System and UI toolkit.

Gosling: Yep.

Hsu: And that was also related to CMU, right?

 © 2019 Computer History Museum

Gosling: Right. So what had happened there was that CMU had got this giant grant/cooperative development agreement with IBM. And IBM built a building on campus and they wanted to build-- I mean it sort of started out as a fancy workstation kind of thing. And so they wanted to hire some people and the theory was that half of the people who worked at this institute would be CMU employees and half of them would be IBM employees. So I became an IBM employee. Technically, my manager was at IBM Yorktown and I spent some amount of time at IBM Yorktown but not a huge amount because the project that we were building which was like a complete workstation, although, initially the workstation itself didn't exist. In fact, I had left the project before the workstation came into existence. We started trying to prototype it on an IBM PC which was comically awful. And, you know, I managed to convince them to buy us some Sun workstations and those were just like gloriously great. And I ended up sort of consulting with a bunch of the groups at IBM. So I spent way more time on airplanes than I was really comfortable with going to principally Austin, Texas and San Jose where two of the major groups were. The group in Austin was building the workstation itself. And the group in San Jose was building the displays. And I kind of got into some of that through my connections at Yorktown because I had actually independently interviewed at Yorktown and one of the projects there that I thought was really cool was this microprocessor called the ROMP. It was a RISC machine. And I thought it was just the coolest thing ever. I spent a bunch of time talking to people who had built the ROMP. So this workstation was supposed to be this fusion of the ROMP processor, the rest of the system built by the folks in Austin and the display accelerators and controllers and all of that built by the folks in San Jose. And so I ended up visiting all of these folks over and over and over again which was mostly an exercise in figuring out how comically bizarre a giant corporation could be.

Hsu: And this predated the RS/6000 series?

Gosling: Yeah. I think it eventually fed into that but yeah.

Hsu: So you had not been involved with the Andrew Window System back at CMU at all? Or had you been…

Gosling: I was the guy who wrote it.

Hsu: Oh, you were the guy who wrote it. Okay. So that was yet another one of your projects that you had done.

Gosling: Yeah. I was the guy who originated it and built the first versions of it. For the first year or so I was the only person working on it.

Hsu: And was that another one of your attempts to do a dissertation out of that work?

Gosling: No, no, no. That came after I was graduated.

Hsu: So you were working at CMU for bit?

 © 2019 Computer History Museum

Gosling: Yeah. So I graduated like right at the beginning of ‘83 and I only worked for IBM for a year-and-a-half before I went aahh this is crazy because it was really clear that their RISC-based workstation was going on a completely idiotic direction and there was nothing I could do to get them to do anything other than drive the bus into a brick wall. So then after a year-and-a-half I went to Sun which is kind of where I should've gone in the beginning but…

Hsu: What exactly was the problem with the IBM project?

Gosling: So it wasn't so much the IBM project itself but everything kind of around the IBM project. So one way to view the problem with the IBM project was that it was way too successful. You know, the ROMP processor was really, really good. And, you know, based on all the early work it looked like they could build and sell for a tidy profit a workstation for $10,000 that probably would have outperformed their $10 million mainframes and that was kind of the core of the problem that all the mainframe folks-- every time we had reviews of this thing the mainframe folks would come up with ways to cripple it. And it was mostly these kind of ways of putting requirements on what standards they followed and what kind of interconnects they did. So while the early designs had all I/O going through some kind of a DMA channel, through some path that I do not understand they ended up creating this requirement that the disk drives had to go through a serial channel. So they sort of forced a lot of PC compatibility. But PC compatibility in ways that made no sense at all. So I was there when the very first units were delivered and I was kind of horrified that you could DMA from the mouse but you had to do a sort of programmed I/O which is like byte-at-a-time for the disk. And when I sort of dug into how that came to be I was like “you guys are so fucked.” And just putting it together how the politics around the project had so crippled the project. So it was like this isn’t my fight. I'm never going to budge this one.

Weber: But most of the problems were hardware and I/O, not the window system? You were given more freedom there.

Gosling: Yeah, because nobody had any clue what a window system was really. The average concept of what a display was in IBM was those 3270 displays. And, you know, the kind of stuff that the Andrew Window Manager could do just did not make sense to them, so they didn't actually care much because it was just too weird.

Hsu: So we talked a little bit about you joining Sun last time, getting recruited by Andy [Bechtolsheim] basically.

Gosling: Yeah. And then saying no, the ROMP will kill you. And then discovering that oh, IBM killed the ROMP, so no worries there.

Hsu: When you got to Sun you essentially worked on yet another window system, NeWS. So was that just a natural thing to do?

Gosling: Well, when I was hired, I wasn't actually-- for the first few months Bill and Scott had sort of said, you know, “Poke around and look for a project, figure out a project to work on.” One of the things that I

 © 2019 Computer History Museum

had a hard time with at Sun, at the time, was that the OS-- and they were just barely starting to use BSD. It didn't do paging. It was this sort of early-- it was still sort of an early version of the UNIX kernel that would just sort of block page entire processes. And a few other things and then the window system-- and they had a window system group and I wasn't exactly a fan of the way it was being built if only because for a variety of sort of peculiar reasons I had built the Andrew Window System using the network. And I originally did it because you could only have one process. Unless you could edit the kernel you could only have one process accessing the display at a time. And so I did what I thought was originally a fairly disgusting hack, mainly I forced all rendering to be done by one process and then the other processes communicated to it over the network. After a while I sort of decided that that was actually a good idea, not a bad idea and it worked really well. So when I got to Sun, the window system that they had ran in the individual application, so there was a copy of it in each application and they went through a pretty complicated dance to sort of switch ownership of the various hardware registers back and forth because the hardware didn't understand multiprocessing at all. You really just had one process at a time. And that had its own issues, too. But it also meant because there wasn't paging there were multiple copies of the same code using up memory and memory was really precious. A machine with a megabyte of RAM was a big one.

Hsu: What year was this?

Gosling: This was ‘84.

Hsu: Eighty-four. That was the year you started at Sun?

Gosling: Yeah. So I decided that I would-- and, also, Bill [Joy] was really eager to have me do the window system thing so I did. And that turned into the NeWS window system which actually worked remarkably well but then there was this horrible industrial politics with DEC and company. It was quite bizarre.

Hsu: And you worked with Warren Teitelman?

Gosling: Yeah.

Hsu: On the NeWS project?

Gosling: Yeah. Well, he was the manager of the sort of window system and graphics group. And the group was sort of two parts. One was the part that had been doing the window system that was there when I arrived. And then Bill wanted me-- was encouraging me to do this other thing so we actually had two window system projects. So for the first year or two I was working alone on NeWS. And then there was this other group and Warren sort of managed both of them.

Weber: But why have two?

 © 2019 Computer History Museum

Gosling: Why not? <laughs> Well, the other one was there, had been there for six or ten months before I got there. After seeing what I had done at Carnegie Mellon, Bill became convinced that they were kind of going in the wrong direction and so he kind of set me up as the sort of competing counterargument. And, yeah, it was weird.

Hsu: What was exactly your line of reporting? It sounds like you were kind of brought in almost at a very high level by the founders of the company.

Gosling: Yeah, I mean, I knew them all. And like I think I said, the last time I had worked with Andy before. But as John Gage used to say, “Sun doesn't have an org chart. It has email.” And, you know, it was-- an awful lot of the early people at the time, which very pointedly did not include Scott [McNealy], were kind of hippie commie weirdo freaks who didn't believe in org charts. Scott was kind of a straight-laced kind of guy for that but he kind of went with the flow. So, yeah, the hierarchy was a very weak notion at Sun.

Hsu: So it wasn’t like you were in research or you were in a product group? You could've been working on anything.

Gosling: Yeah. So when I was first-- when I first graduated and I was trying to figure out what my first job would be I had this thing in my head that I really wanted to work in a research lab. And for most of my time at Carnegie Mellon I had really wanted to work for Xerox PARC. But then I get towards the time when I'm graduating and Xerox PARC was just falling apart. And I interviewed at Xerox PARC sort of before it completely crumbled. But by the time I was graduating it was pretty much gone but I still wanted to work at a research lab. And so when Andy approached me the day that Sun was founded and IBM was twisting my arms and they had this ROMP processor that was clearly way better than anything that Motorola was building, I kind of went with the research lab because that's what I’d always wanted to do. And then after a year-and-a-half of figuring out what a straitjacket the research lab at IBM was wrapped in, I sort of realized that Sun was more like a research lab than any of the research labs. So I went to Sun. And for the longest time Sun didn't have an R&D department, a separate research lab but that was okay because most people at Sun were nut cases in the best possible way.

Weber: Did you look at DEC at all? A lot of PARC people went with Taylor to DEC and stuff.

Gosling: Went to DEC? Yeah. No. I guess I had gotten religious about the whole UNIX thing by then because I had been involved in some of the VAX VMS versus UNIX wars and that involved doing a bunch of valuations and work with VMS and I was like, oh Lord, no. I don't want anywhere near that stuff.

Hsu: Why did X Windows beat out NeWS in the market?

Gosling: <sighs> Personally, the amount of money that whole consortium put together that heavily went to marketing was just completely overwhelming and several people at DEC had said that when NFS first came out DEC had a network protocol that they were using for their equivalent of a file system. And it was crap. And NFS won. And several people from DEC had said, “We’re not going to be NFSed again,”

 © 2019 Computer History Museum

because NeWS was starting to get pretty popular. And they thought that this was completely awful. So they actually-- when they sort of announced the X consortium pretty much every player in the computer industry was there and they held this press conference and they actually had security guards barring people from Sun. We had sent some people, I wasn't one of them, to try to attend this press conference so we could at least find out what they were saying. And anybody who said that they were from Sun, security guards kept them out. And it was like, really guys?

Weber: And that was Michael Dertouzos, right,…

Gosling: I don't know who was…

Weber: …running the X consortium.

Gosling: Yeah. I don't know how any of that stuff happened. I just know that the couple of people that we had sent to try to attend the press conference were physically restrained from entering.

Hsu: With NFS and NeWS it seems like very early on Sun was already branding itself as not just a workstation company, not just a UNIX company, but also a networking company.

Gosling: Oh, the networking was from day zero. One of the things that really attracted me to Sun was one of their foundational bits of religion was networking. And I had come to really believe that networking was a big deal. And, you know, IBM clearly did not believe that networking was a big deal. They had this evolving communications standard called SNA. It wasn't a standard. It was a basket of random bits of whatever they just sort of put together. It was, you know, as me father would say “a dog’s breakfast.” And DEC had this thing called DECnet which was not quite as lame as SNA but it wasn't nearly as-- it wasn’t anywhere close to being as elegant as the whole TCP/IP suite, which had a pretty strong ancestral relationship to all of the stuff from Xerox PARC, the PUP/BSP and TCP/IP were very, very close parallels.

Weber: But XNS was still in play at this time, right?

Gosling: Yeah. But that had left PARC really and it kind of fell apart. And that was more-- that wasn't so much a technical comment as it was Xerox trying to milk this thing for money. And one of the things about TCP/IP was that there wasn't really any commercial interest in charge. That it was all about getting systems to talk together. And since, you know, the systems that talk are just a vehicle for people to talk at a level that included more than just voice.

Hsu: So the NeWS project, at what point did your work on that finish?

Gosling: <sighs> It sort of took a bunch of just really weird turns but I was pretty much done by ‘89. I just couldn't take it anymore.

Hsu: Between that and the start of the Green project were you working on anything else?

 © 2019 Computer History Museum

Gosling: I was trying. Nothing really went anywhere. There was about a year of floundering.

Hsu: We did talk about the beginning of the Green project last time. I sort of want to drill down a little bit more into sort of Patrick Naughton's primal scream email.

Gosling: Yeah. So he had-- because he was one of the people that was working on the NeWS window system and a bunch of us were really, really disenchanted. And he had interviewed at NeXT. And he was really disturbed that these guys were doing something really interesting. And, you know, the fact that they were sort of trying to do something that was kind of out of bounds from the normal computer industry. And I can't exactly remember that first email but it was kind of like, “Sun is missing out.” And there were a bunch of us who were kind of like, yeah. But Scott [McNealy]'s reaction was kind of interesting which was essentially to say-- and so myself and a couple of other people, Ed Frank, Mike Sheridan kind of joined in this sort of this just doesn't seem like we're doing the right things. And so Scott said, "Well, why don't you just take some time and figure out what the right thing would be." And so we did. And Patrick decided to not leave-- because I was kind of like floundering and trying to get stuff started. And Sun had become a little more political than I was really happy with. So we went off in a corner, we did a few road trips to talk to people who were building sort of systems that were more tailored for actual humans. And that sort of drew us into conversations with a lot of people who do consumer-electronics and the way that things were being-- digital systems and consumer things were being sort of integrated together. And then we decided to start to try to explore the space and since we were engineers we started trying to build prototypes of things and we got a bunch of people to join us. This sort of manifesto got written, I think, mostly by Mike with Ed and everybody else sort of piling on but Mike held the pen. It was this weird document called “Behind the Green Door,” for all kinds of reasons besides the fact that the little pod of offices we were in had a green door.

Weber: Besides the movie, what would the other reason be?

Gosling: The green door. We had a physical green door and Apple had this new operating system project going on that was called Pink. So we figured that if we’re going to needle them we might as well pick something with a color in it. And our office space had a green door and then there's the obvious movie reference and it was just compelling.

Weber: You probably joked about it.

Gosling: Oh, yeah.

Weber: You’d talk on the phone, “Are you behind the green door?”

Gosling: Yeah.

Hsu: How familiar were all of you with NeXT and what they were doing?

 © 2019 Computer History Museum

Gosling: Somewhat. We had bought a couple of NeXT machines and had played with them. You know, in my time of floundering one of the things that happened was-- because I was still doing work on things like graphics device drivers and that, was that the first NeXT machines had this weird grayscale frame buffer that had-- it was a two-bit frame buffer so that you got four pixels in a byte but it gave them four levels of gray. So it was a way of getting, you know, more colors but only a smaller number of-- sort of, without using too much RAM for the frame buffers. So there was some hardware engineer folks at Sun that sort of went, you know, that's actually really easy to do. It's like one extra chip in the frame buffer. And so they actually made a two-bit frame buffer and then said, "Well, let's ship it," without having thought at all what it would take to do the software to support a two-bit frame buffer. And because of what it takes to build all the rendering software, a two-bit frame buffer was a big problem. It was going to be a huge amount of work. It was just going to be huge. And, you know, I blew several months just on the warpath, you can't release this frame buffer. Yeah, sure. It does the NeXT coolness thing in the frame buffer. But there's at least ten man-years of effort here. You either fund us to do the work or you don't. So far you're not funding us so the work is not going to happen. So how can you release this frame buffer with no software? So I blew damn near half of that floundering year was just fighting that battle. And, yeah, and a lot of the main graphics folks at Sun, because of the way that X11 had worked, what that really did was it caused Sun's window system engineering to just seize up because we were trying to sort of respond to the X11 thing, in retrospect, mostly in ways that didn't work very well. And I was pretty ticked at the way that that had sort of unfolded. And so we just sort of left that all behind and we just were looking around and kept finding all kinds of interesting stuff sort of out there at all of these consumer companies. And it was this combination of interesting, cool and completely horrible because, for example, they were all reinventing networking from first principles and making all of the mistakes that people in the computer industry had made only, on average, doing appalling things. It was stuff that was-- people were doing stuff that was known to be stupid in the sixties.

Weber: Give one example.

Gosling: One of my favorites was a network protocol that the network protocol was meant to go over a serial bus kind of like RS-232. The packets had really no error checking at all. And they had an eight-bit address, so not going to work even close. And, you know, I tried pointing out that serial lines they lose bytes. Bytes get corrupted. You need to have some kind of integrity checks. And, you know, just talking with these folks was just really difficult because they couldn't see why anybody would need more than eight-bits for an address because you're never going to connect networks together and networks are never going to be big. It was like, yeah, wrong.

Hsu: I think it was really interesting that Scott McNealy was maybe enlightened, maybe so to speak, to sort of let you guys just go off and start the Green project. I guess it was either that or lose you.

Gosling: Yeah. And, you know, we were-- we argued pretty hard about why Sun was screwing up. And Scott was never a technology guy. As he describes himself in college, he majored in golf which, I think, is kind of unfair to Scott but that's the way he would describe himself. <laughs> But he did have faith in people. And, you know, if the four of us said that Sun was screwing up then maybe there was something there. So he was, like "Yeah, go take a stab at it." And this was a really vague primal scream because we

 © 2019 Computer History Museum

couldn't-- early on we were having a hard time really articulating what was going on. So we went on these road trips and came up with a plan and 90 percent of the plan failed and 10 percent succeeded.

Hsu: Earlier you mentioned there was green-- the color thing was kind of a response to Apple's Pink. How much were you paying attention to what Apple was doing?

Gosling: Not much because it was a deep dark secret. The folks at Apple are really, really good at keeping secrets.

Hsu: I did feel that the UI of the Star7 kind of reminded me a little bit of HyperCard and General Magic. Or was that just sort of an aesthetic that was out there?

Gosling: Well, yeah, we were very much aware and thought about HyperCard and that. For a while we were sort of thinking that that was kind of the way to go. But if you looked at the scripting language behind HyperCard it was pretty limited. A lot of the things that people had been wanting to do were a little more complex. And even though it was from Apple and all of that it was still-- still built user interfaces that were fairly unfriendly. So we were trying to really overachieve on the friendly side of things. We went full cartoon. And kind of as a way to be just as far out on the crazy edge possible. And it really was too far out there. Certainly, the performance requirements of building an interface like that was, you know, if you've watched the videos they're all kind of jerky and that. And that was a pretty wimpy processor that we had back then. And a certain amount of the HyperCard kind of thing of going from frame to frame to frame when you've got a small screen it's kind of hard to do anything else because you can't have multiple windows on the screen that's that small. There's a limitation to just how much animation and how much of the dynamic behavior you can do. And we really wanted to have things that did not look computery in any way. So instead of having buttons with boxes around them to signify that this is a button, we did it with saturated colors versus more muted colors. So things that were more saturated were things you could press on. And things like that actually worked out pretty well.

Weber: Walter Smith of the Newton project actually said, just the other night, that he felt NewtonScript was quite close to Java. But you didn't know--it's not like you knew what they were doing at that time.

Gosling: Yeah, we didn't know that.

Weber: But there was the-- given that you were moving toward small devices, Internet of Things, for want of a better word, that was the same period, obviously, that there's Newton, there's Go, there's General Magic. How much of the others were you looking at? And why didn't you think of going in a more, you know, the device being your main interface like those folks did?

Gosling: Well, because the thing that was really pushing us was not so much the device but the interaction of the parts of the ecosystem. And the way that Java was designed, one of the-- it had a bunch of really important influences. One was like if you're on a network and you're amongst consumers you have to be really, really serious about security. And then there are a whole bunch of architectural follow-ons that come out of security, things like if you looked at most of the security failures in the Internet at that

 © 2019 Computer History Museum

time and it's still the same now. Most of them are due to bugs and most of the bugs are things like buffer overflows or null pointer exceptions. And the fact that in Java you cannot turn off like array subscript checking and you cannot turn off null pointer checking and you cannot violate the boundaries of an object. Those entirely come out of security considerations. And so the cracks that come in through those things which are just common these days, they can't happen. And then from my experiences with the VAX PERQ translator it was like yeah, if you design the language right you can get both semantics that feel like a pretty high level language and, yet, get really good performance. So things like a strong type system and being careful about what do numbers mean and yada-yada, you can get the kind of-- a lot of the feel of some of these scripting languages without giving up much, if any, performance and without giving up-- trying to make things a secure and reliable as possible. One of the things that just came out over and over and over again with the consumer electronics folks was-- rule zero was effectively "thou shall not kill your customer." And in the computer industry, particularly in places where Sun was competing, it was all about performance. And if you could trade a little bit of reliability for a significantly large boost in performance you would take that. Certainly, companies like Cray did that all the time, I mean, with the way that they did their floating-point divide, division by Newton's method so that the bottom bits weren't always particularly accurate. Yeah, that just didn't seem like the right trade-off.

Hsu: You mentioned you went full cartoon. Could you talk about Duke? Were software agents, you know, the idea being kind of en vogue at the time, you know, Clippy .

Gosling: Well, Clippy came after by a couple of years.

Hsu: Well, Knowledge Navigator, I guess.

Weber: And General Magic with agents.

Gosling: Yeah, so, yeah, and we didn't see any of the General Magic stuff until afterwards but particularly Mike felt that since this was a consumer thing and people were used to interacting with people, not machines, that to have something in their mental model that they were interacting with that was not mechanical, that was more of a person. And we had this artist working for us, a guy named Joe Powrang [ph?]. And his history had been all cartooning. I believe he originated the Scrubby Bubbles character. And he did a whole lot of different workups for possible designs. And one of the problems we kept having was that the screen was really small and Duke needed to look pretty reasonable when only 50 pixels high so there's not much room for complexity. And so Joe spent a lot of time on coming up with this really minimal shape that could still show emotion. And we get a lot-- so from the nose you kind of intuit where the eyes are. And since the top-knot sort of gives you head angle he could project a lot of emotion with what was essentially just a triangle with a red dot. And so we went with it. I think there were some of the designs he did that I, from an artistic point of view, I'd like better. He did a really cool robot character, sort of a sad sack robot that was, I guess, sort of what's his name in "The Hitchhikers Guide to the Galaxy"? Marvin. But this was kind of like the happy version of Marvin. But the problem was that there was no way to draw him small and make it come out right. And Duke worked really well, and Duke had this kind of really nice, friendly aspect. There's the Disney code that anything with three fingers and a thumb is a cartoon character. And if it's got four fingers and a thumb then you're trying to be realistic because Mickey Mouse

 © 2019 Computer History Museum

always has three fingers. And whenever they were trying to do more people that were trying to be not cartoons but humans in a scene they always had four fingers which I didn't know until Joe explained all this to me, these little hidden codes in things.

<01:04:20>

Hsu: So we were getting into sort of talking about the language. When FirstPerson started you hired a number of people from NeXT?

Gosling: Yeah. Well, so when FirstPerson started but from the Green project there was nobody from NeXT in the Green project. But then when it went through that NeXT-- because by then NeXT was getting into trouble. They weren't taking off the way they needed to.

Hsu: Could you talk about the interfaces in Java and how much was that influenced by protocols in Objective-C?

Gosling: Very heavily. So classes in Java were kind of influenced by classes in C++ but mostly they were influenced by classes in Simula. And one of the things-- because I had used Simula pretty extensively including having spent way too much time pawing through the compiler or at least the original NDRE Simula-67 compiler. And one of the things that really thrilled me about the way that they had done it was that for Simula-style classes there was this great trick for making them really fast to do method-dispatching, the virtual function table, which oddly enough, were the first written example of a virtual function table was in Ivan Sutherland's Sketchpad thesis. But, you know, it's a grand and simple technique. And then as-- one of the things that I had a hard time with in C++ was the whole multiple inheritance thing because it had made it very difficult to do efficient dispatches. And one of the things that I liked about having interfaces as something separate from classes in, essentially, the Objective-C style was that it gave the language a hook to distinguish when it could use high-speed dispatches versus the somewhat slower lookup table kind of dispatches. And as time progressed the compiler technology got to where actually almost all interface dispatches could be made as fast as class dispatches. But it requires some fairly ugly analysis which on today's computers is straightforward and not a big cost. On machines at that time where a megabyte was a lot of memory, yeah, that was kind of above and beyond. But it was also a way to sort of-- some of the issues around multiple inheritance get kind of messy, whereas with interfaces they’re a little bit cleaner. And so we had hired some of these NeXT folks and they really, really, really, really liked the Objective-C technique. And I was literally getting people walking into my office all the time and I was trying to be like how can I make these guys happy because they really have a point. And it started to feel to me like, you know, these kind of were different universes, the sort of interface style and the class style and rather than try to mush them together in the sort of multiple inheritance techniques in C[++], I just sort of decided, well, it's actually pretty clean to keep them separate. So that's what I did.

Hsu: So that was less driven by your personal experience programming in Objective-C and more by these NeXT folks that had come in.

 © 2019 Computer History Museum

Gosling: Yeah. And they had really good arguments. So I listened and I tried to figure out how to sort out both sides.

Hsu: What were their names?

Gosling: I can't remember.

Hsu: Were there any other sort of things that they brought with them from NeXT or Objective-C that got into Java?

Gosling: The interfaces were really it.

Hsu: Talk about the influence from Mesa.

Gosling: Yeah, I had actually used Mesa a moderate amount when I was in grad school.

Hsu: Is that because of Bob Sproull?

Gosling: Not so much. But the fact that Xerox had made a grant to CMU of a couple of dozen Altos and stuff and so I just played with them. There was all this stuff on the Altos and, unfortunately, the problem with the Altos is that there weren't really enough to go around so it was hard to do a serious project on the Altos because they were a scarce resource. But I could build stuff in Mesa and play with it. I kind of liked the way that a basic threading model was built-in to the language and how it made life cleaner. And when I had used Simula, Simula also had coroutines built into it which felt kind of like the threads in Mesa but in Simula the threadlike things really were coroutines and so there was no asynchrony. Whereas in Mesa there was asynchrony. And one of the interesting things at Carnegie Mellon in that era was that there was a lot of work on multiprocessors and I had been involved in that. So I was in this kind of ethos that was all about parallel processing. And the way that they had done some of it by sort of building hooks into the language really cleaned things up. And they made it feel more like sort of the multithreaded aspects that appeared in Simula were great but coroutines depend on there being just one core. If you try to do coroutines on multiple cores you get all of these concurrency issues that just get really complex. The coroutines are attractive because you don't need locks and things. You can just do parallel processing and because you know in context what just happened you can say A = A + 1 and you don't have to worry about concurrent modifications to A or things like that.

Hsu: You mean threads?

Gosling: Threads, processes, pick a term.

Hsu: You're saying threads, not coroutines are good for concurrency but coroutines were made for one processor…

 © 2019 Computer History Museum

Gosling: No, no. So the kind of concurrency that happens in coroutines is where you do explicit switches. And you program in a style that feels like it's multiple threads but really it isn't. So you can never get truly asynchronous behavior so if you've got-- in a coroutine if you've got a statement you know that nothing is ever going to run at the same time because the only time you switch from one thread to another is with an explicit yield. So you kind of get locking for free. But the price of that locking is that you can only run on one core. So things like Golang had a really nice concurrency story when they only supported one core. And then as they sort of supported multiple cores then they had to pull in all this stuff about having a very complex memory model so you could talk about synchronization issues across all of these cores with all of these memory issues. And I thought that Mesa had done a pretty nice job of it. So I decided that since multithreaded processing was just the right thing to do that it just needed to be built in. And by having language support it got a lot cleaner.

Hsu: And Jim Mitchell who had worked on Mesa at PARC had, at some point, joined Sun and helped out on Java? Or was that later?

Gosling: Yeah, that was quite a bit later. Yeah.

Hsu: Another thing you mentioned, and I don't know if this was because of Mesa but the static typing, well, you did mention in the context of security. So it's interesting that Mesa and then Java both had static typing but dynamic binding. I think that's a very interesting combination.

Gosling: Yeah, well, so one of the lessons to me that came out of the PERQ Pascal translator was that the byte code-- so if you look at something like Smalltalk that does not have strong typing, because they didn't really care about performance, it was not a really big deal but-- because I had read a bunch of the papers on Smalltalk but at that time, I hadn't actually used it. But then the Pascal which was a strongly typed language and the type system percolated through to the bytecode so there was like an integer add and a floating [point] add and all the rest, the thing that impressed me about that was that it let me generate really, really tight VAX code and the fact that I was generating better code for the VAX than the C compiler, of course that was the C compiler of the early '80s, actually I think it was the late '70s. So that wasn't much of a C compiler but still, I could beat it pretty handily and all of the reasons that that worked was the static typing and the fact that then it gives you a way to do all kinds of analysis and I was actually doing a certain amount of global flow analysis on the bytecodes to put together the VAX assembly code and I thought it was just really slick that I could make things that fast.

Hsu: I find it interesting that one of the sort of critiques of Java versus more dynamically typed languages like Python and Ruby is this idea that the static typing makes Java code a bit too structured or not as flexible as something that you could do in Python, what would you say to that?

Gosling: Sort of yes and no, I mean personally I like to reason about programs and I don't like debugging, I like stuff to work and one of the nice things about static type checking is that it can give you information about errors earlier and so the static typing gives the compiler a lot of tools to use in analyzing your code so that if you-- it's unlikely that people would really want to add one to a string and you can

 © 2019 Computer History Museum

totally do that in Python and if you say "Bob"+1, it'll try to parse Bob as an integer or will it try to promote 1 to a string?

Hsu: Or is that string concatenation, the + is overloaded to mean string concatenation or something?

Gosling: Yeah. So I always felt that the weak typing stuff made things a little more wishy-washy, it was less easy for the compiler to discover errors and it was just way harder to build high performance software, and if you don't care about performance then okay. So the way that you can-- if you look at something like JavaScript which is where so much work has been done in the high performance implementations of them, those things are really complicated, to make JavaScript fast is really, really hard and you take variations of JavaScript like TypeScript, so TypeScript is just adding types to JavaScript and then that lets you build a runtime that is way faster and much more deterministically fast. One of the arguments early on was, "It's quicker to type your JavaScript code," it's like, really, you're still typing “var” so why not allow some alternate spellings of “var” like “int” and “flt”, I mean if you don't like doing names longer than three characters, because JavaScript would go so much faster, if you could just do the primitive types and then the way that objects are done as essentially hash maps rather than-- you had to do a hash table look up unless you've got one of the very spooky compilers like the one in Chrome, it's like one load instruction, you type p.x, that's one load instruction in Java and it's a hash table lookup. So it's a big deal and some people care about fast.

Hsu: Some have argued that a weaker typing system can make programmers more productive in the prototyping phase or in the…

Gosling: Sure. But-- and so I'm a big believer in prototyping but my measure for productivity isn't from start to when the first prototype starts to run, I actually care about the production quality code which is why I-- I mean I care about doing as many checks as possible, I really care about checked exceptions. I mean I lost a lot of time in my life over the way that so many things in C-- you can just ignore errors and, "It works on my machine," when you say, “fd=open” of some file that contains whatever and, "It works on my machine," and then you ship it to people and it falls over miserably but it falls over miserably in mysterious ways because you say “fd=open” something or other, the open just returns -1 or 0 and then it's not until way, way later that anything actually goes wrong even though the source of the error was right here. I want to know the actual source of an error, when did things go wrong, I mean I don't care, I mean I slightly care about how long it takes to get the first demo to work, [but] what I really care about is how long it's going to take until something actually useful works. I mean when you're doing like a school project, it only has to work once but when you're in a more commercial project, commercial context, it has to work every time and things like type systems and checked exceptions are really useful to help you get to that sort of level of it'll just work.

Hsu: Yeah. Can you talk more about the exception handling?

Gosling: So once upon a time, so when I first implemented exceptions, there were no typed exceptions and that was one of these ones where I kind of did-- because part of the thing at the time was to build something that looked familiar to C++ programmers, people who knew C++ so that they could just look at

 © 2019 Computer History Museum

Java code and understand it, so exceptions were, they kind of worked the way that they did in C++. But Bill Joy really, really felt strongly about checked exceptions and there was a point where he and a guy named Arthur Van Hoff, they literally came in in the middle of the night and implemented checked exceptions and then I sort of went from being, "Why? This sounds stupid," to being a believer and in the end I really liked checked exceptions because I just kept finding that they were just saving my ass all the time. And now I've become pretty, I mean I'm remarkably religious about exception handling these days.

Hsu: Can you talk about adding garbage collection?

Gosling: Garbage collection is something that I had believed in from the dawn of time and when it comes to security issues, I started thinking about garbage collection when I was thinking about security, this may sound a little bit strange but in the C style of doing storage allocation, when you free a pointer, you have to free a pointer at exactly the right time, right? If you free it too late, you're wasting memory, if you free it too early then somebody's going to use this thing that has been released and you end up with memory corruption bugs and a grand source of failures and those sorts of failures get particularly bad when they start being exploited as security vulnerabilities. And so it wasn't so much about allocation as it was about freeing. And so if you just leave free out and you use a garbage collector, A) the whole system gets a lot more reliable, B) it eliminates an awful lot of memory leaks and C) it's just easier for developers. And at the time, garbage collection kind of had a bad rap because garbage collection was kind of associated with Lisp and the universe was very divided over whether or not Lisp was a good idea or an agent of the anti-Christ, I mean it was just incredible how people were divided over it and I had written a lot of code in Lisp at the time, like my whole PhD thesis was written in Lisp.

Hsu: Not a statically typed language.

Gosling: Not a statically typed language although the Lisp runtime that I was using did allow static typing so you could declare things as fix nums and flow nums and the compiler would actually use that and doing fix num declarations in particular at the right times. I went through a bunch of processes of tuning my thesis project and fix num and flow num declarations at the right places made a humungous difference. But I sort of figured that by the time I graduated I had pretty much used my entire lifetime quota of parentheses and the whole bit of garbage collection, it just made life easier. So when it came to Java, it solved some real problems and made life easier and if you can sort of break people away from associating garbage collection with Lisp, it actually turned it into a useful thing so that even though garbage collection had been around for a long time, it had never really gotten much in the way of mainstream adoption. People have done all kinds of studies that showed that like a good garbage collector beats malloc and free pretty substantially, people were still sort of reluctant but with Java, it didn't really make a big deal of it, it just sort of snuck up on people, you would just say “new foo” and don't worry about free and then people just went, "Oh great." And then it was like, "Oh crap, you mean I have to use free when I go back to C?" That felt to me like a real sleeper hit.

Hsu: But given that you were initially targeting Java for these small consumer devices, was the nondeterministic nature of the collector, did that have any performance problems?

 © 2019 Computer History Museum

Gosling: No, I mean malloc and free have pretty strong non-determinism too although people mostly don't think about it, if you look up the code for malloc, I mean it still does searches, free does all kinds of weird stuff to coalesce things, and they're not simple either. And the issue with a garbage collector is often with the fact that it sort of takes a lot of these overhead chores and instead of having like low level noise, it just has a spike every now and then, but on small memory machines, the spike is really teeny, certainly on the original Star 7, the garbage collector pauses were never noticeable. But what was a bigger deal was we didn't have memory leaks and on a small memory machine, memory leaks are a big deal. And it was a fairly primitive garbage collector in that one but it did also do storage compaction. So if you can eliminate fragmentation and leaks, that's a big, big deal in a small machine.

Hsu: We've already been talking about security, what other sorts of security features were important, sandboxing was one of the…

Gosling: Yeah, sandboxing was a really important thing particularly early on with applets, the fact that you could take a piece of code, surround it in a sandbox and have everything that sort of pokes out of that sandbox get inspected, it fundamentally worked pretty well. In the early days, applets had this real problem that there were all kinds of security issues but the thing that was sort of annoying about them was that the sandbox was never the source of the security issues, there were pretty much two flavors, one was doing anything with the word security in a sense on Windows particularly in the era of Win 95 and Win 98 was just crazy, the things we had to do to make up for that were just crazy. And the other was that, mainly the issues were sort of leverage attacks, so early on we did things like trusting the domain name server and we would use-- so for sort of same origin security checks we were using the DNS, the problem is that the DNS can be attacked and a lot of DNSs were pretty insecure. So people came up with these techniques where you could corrupt a DNS and then Java was more trusting of the DNS than it should have been and so then you could use that and you could punch a whole, so we became increasingly paranoid about DNSs but I mean DNS corruption is a huge issue and it's been pretty thoroughly addressed, it's been a couple of decades of, or three decades of people hardening and hardening and hardening the DNS. It is sort of sad that many of the current vulnerabilities in the Internet are people having been sort of shortsighted 30 years ago, I mean the fact that a lot of email spam is just because the SMTP standard isn't nearly as tight as it should be, there's no authentication really, when you get an email that says it's from whoever, you don't actually know if it came from there, there's no-- and there are authentication extensions to the mail transfer protocols but they're essentially unused and I've never figured out why they're unused, but you can get a certificate. Amongst some of my paranoid friends, they all have certificates and they delete mail that doesn't come from a certified source but that means if you buy a ticket on Ticketmaster, Ticketmaster sends you an email and you have no idea whether it actually came from Ticketmaster or somebody spoofing Ticketmaster and they'll send you a fake ticket to Pink Floyd or whatever and it's like, "Great, here's your 50 bucks, that was a deal."

Hsu: So I want to go back to sort of the-- because Java sort of started off as kind of like being a better C++ and you had mentioned you had had a lot of experience with Simula, the object-oriented part of it was always assumed from the beginning, there was never any decision to not make it object-oriented?

 © 2019 Computer History Museum

Gosling: Yeah, I mean I had built object-oriented stuff from the dawn of time. So one way of viewing Java is that it was my way of taking a lot of interesting high tech ideas, dressing them up in a way that electrical engineers would understand. So like garbage collection, it was there, never made a big fuss about it mostly because that would scare people away, object-oriented was a bit of a leap for people but I mean that was a really deep and abiding thing for me. Unicode was strangely controversial and I was like super religious about it because it was just the right thing, I was sick and tired of dealing with all these different encodings and I wanted to make it so that it wasn't so crazy to write software that worked in the connected world. If you've got a network and the network spans the planet, you got to be able to send email-- my grandmother died a number of years ago and I wrote up the little blurb that you hand out at funerals for her and there was so much stuff in her family history that you couldn't write in ASCII, right, and I'm not exactly sure-- well I guess that was before Pages really existed on the Mac, anyway I had to write it in Microsoft Word and one of the characters always caused Word to crash. Oh, I remember what it was, yeah, I'd gotten a bunch of material from the Icelandic Consulate and when I imported it into Word on the Mac, because it was a Word document from the Icelandic Consulate that had-- because for reasons I don't really understand but if you live to more than 100, the Icelandic Government does an official write-up on you and they sent that to me as a Word document which I brought into the Mac Word document and I loaded it up and I discovered that there was one character that if I deleted that character then Word would not crash. And it's like <deep sigh> and I forget which one it was but there were all of these weird runic characters in her real name and the name of where she came from and I was like, "It should not be this hard." So Unicode.

Hsu: Had you been a part of the object-oriented community and going to OOPSLA and these sorts of conferences?

Gosling: Not much because I was never really a programming language person, I did Java not because I was a programming language person but because I had a stack of problems and I kind of ended up going-- the programming language is part of the problem and I can solve a bunch of these problems by just doing yet another programming language. I mean I had implemented compilers in the past but never thought of myself as like a complier guy, I mean I had done all kinds of stuff and I was more a do what gets the job done kind of guy.

Hsu: And Java is a pure object-oriented language so everything has to be a part of a class, in an object…

Gosling: Well except for the primitives, right, which is where lots of people get cranky.

Hsu: But ints and things are still objects aren't they, integers?

Gosling: No, and-- well so there are the primitive objects like int and then there are wrappers that are true objects so there's an integer class but that's a wrapper for an int primitive. And I expose the primitives because I could not figure out a way to make a true pure object-oriented language efficient enough. Like in Lisp, most Lisp implementations cheat in creative ways so if you try to be truly object-oriented then everything is an object which means that if you've got an integer, it's wrapped in all the type

 © 2019 Computer History Museum

information that is an integer object and you can have a pointer to it but that means that if that's what an integer is, it's something that is allocated in the heap, you can't just have its entire lifespan be in a register. If you do some really exotic analysis you can kind of make it happen but even then, it's really hard to get a number of the primitive operations to really work, so like in Lisp, it will represent integers in somewhat magical ways. So if you've got a pointer to an object and it's a pointer in your address space and it's fairly common for pointers to be on double boundaries which would mean that the address is always a multiple of eight so the bottom three bits are always zero. So a common trick is to use those bottom three bits as a type code and so if you've got an integer then it's the top bits are the integer and the bottom three bits have got a-- this is an int thing so that you can have an integer and you don't actually allocate memory for it and you can do that for seven primitives, seven primitive types because you need one more type code which is zero for the actual object and that particular hack has been used so many times. But one of the things that causes trouble is that then the things like equality can behave a little differently because as soon as you get an overflow, so if you take off three bits then you've got 29 bits left so if you've got an integer or whatever the layout is, you end up having a restricted set of integers that can be represented as these kind of magic primitives. So if you keep adding and they overflow then a real object gets created. And so then what happens is that, if you say A = 2 + 2 in a pure object world, that would mean take this object "2" and this object "2," add them together and create a new object. And in Lisp there are two kinds of comparison, there's = which is compare the value which is kind of like a deep one that follows the pointers and then there's eq which just checks the pointer. So if you write A = 2 + 2 and then B = 2 + 2, in the pure object-oriented way, those "2 + 2"s have generated separate objects so that they should be = but not eq, right, so that the addresses to them are not equal. But if you do that then every time you add 1 to something or you add 2 and 2 to get 4, you'll get like this profusion of 4 objects, so you've got to try to collapse them and so that's what the little hack of using the bottom three bit type codes ends up trying to do is making sure that all 4s are the same, so that if you say A = 2 +2 and B = 2 + 2 then in a Lisp that has this particular hack then it will not only be = to B but it will also be eq to B but if you keep adding to these, they will eventually overflow out of the range that this hack representation works and most Lisps would then start allocating objects and then you can tell a lot about a Lisp implementation by just multiplying by 2 and looking for-- they'll usually start out where = and eq return the same thing and then eventually eq and =, one will return true and the other will return false and that tells you where the caching hack is blown apart. And that gives you this weird view of what like equality means and yet if you don't do that then performance goes to hell, so you either get weird semantics or you get weird performance but you do at least get a-- yeah. So I decided I cared less about object-oriented purity than sort of performance and coherence.

Hsu: One thing, one critique of Java I've read is that sometimes programmers just want a free function and in Java there are no free functions so they have to make everything a method to a class and so they may have to just create an artificial class just as a boiler plate just to hold the function.

Gosling: Yeah, but that, when generics came, sorry lambdas came in like six or seven years ago, that went away. And that was another one of these ones where it took a long time for people to figure out a way to do it without any performance apologies. Even though I haven't been involved in really building or evolving Java in quite a while, one of the sort of mindsets that has stuck is trying to do it without sort of introducing performance problems. And a lot of it turns into fairly rocket science compiler technology.

 © 2019 Computer History Museum

Hsu: So I'm interested to know how, you know, we talked a lot about Bill Joy. How much, I guess he was sort of overseeing the whole project? How much influence did he have and what was sort of his role?

Gosling: He would kind of pop in every month or two, and we would talk and he would respond somehow, sometimes he had things he like really pushed on, and sometimes he didn't. And sometimes I listened, and sometimes I didn't. I mean, we had known each other for quite a while and we would talk. Mostly things were just fine. We occasionally got a little heated, but not that often.

Hsu: What sorts of like specific things would you say were due to his influence?

Gosling: Well, certainly checked exceptions. He really wanted to do generics and lambdas really early on, and I would have loved to, too. But I kind of-- I pushed back pretty hard on it, because the design space was just like gigantic. And the people who had done-- I mean, it was still a huge ground for getting PhD theses. And there was nothing like a consensus for what mechanisms work the best. And if we were going to do that in Java, we would have had to delay everything for about a couple of years. And I was like, "We got to launch. We got to get this going."

Hsu: Right. So he was pushing more for like academic Computer Science-y features, and you were pushing more towards practicality.

Gosling: Yeah, I'm a much more blue-collar kind of guy than Bill is. I like to get things done. I like to build things that don't break. And I like things that are fast.

Hsu: So then let's see. We went over the creation of First Person. Who was in charge of-- who was actually like leading First Person?

Gosling: A guy named Wayne Rosing.

Hsu: Ah, right, Wayne, yes.

Gosling: Right.

Hsu: Wayne had been-- he was already-- he had been part of Sun already before that.

Gosling: Right. Yeah, and as a guy to lead an R&D effort, he was pretty good.

Hsu: So what was-- how did Eric Schmidt's role as CTO affect the development of Java?

Gosling: He was almost entirely hands-off.

Hsu: Okay.

Gosling: He talked about it onstage a few times. And he was in the Org Chart, but--

 © 2019 Computer History Museum

Hsu: Okay.

Weber: What was John Gage's title at that time?

Gosling: I think he always had-- I think he was Chief Science Officer, even though he's one of the least scientific people you've ever met. He is. But he's extraordinarily eloquent and very passionate. And could give and could get scientific audiences, or even non-scientific audiences motivated about just about anything. One of the best public speakers I've ever known.

Hsu: So in 1994, one of your advisors, Guy Steele, joins Sun. And so how much did he end up working with the group to--

Gosling: So he joined kind of later on. And he never really joined because he was always in Massachusetts.

Hsu: Oh, I see.

Gosling: But in, you know, after this whole First Person thing, we had sort of decided that it was-- that was just not going anywhere and we decided to transition over to the Internet. You know, we had decided-- it was part of that hot-tub thing that we would stop trying to get all the phone companies and cable companies interested in doing something. And we just decided just go for the Internet. But one of the problems was that the Java spec wasn't really a spec in any real sense. It was written more like the Kernighan and Ritchie C book. It was more conversational. And so Bill [Joy] figured that we really needed a formal spec. And you know, I told him, I just don't have the kind of brain that can do that and have it come out sounding proper. And so Bill took a stab at it, and he swizzled it into kind of programming language standards, [Backus-]Normal Form. And then he kind of collapsed and went, "I just--," there was just like so far he could go. But Guy Steele was like an absolute grand master at fussing the details in language specs. I mean, he had worked on things like Common Lisp. But so he got-- so then Guy got pulled in, and you know, whereas I had kind of cruised over a lot of corner cases, Guy just has this laser for all the little dark corners, and "what exactly does ‘shift left’ mean when the shift count is greater than something or other? And what exactly does the mod operator mean in certain corner cases? And just exactly how should we interpret some of the waffle-y bits in IEEE-754?" And yeah, so he really went to town. And you know, the thing that most people think of as the Java programing language spec is mostly due to him.

Hsu: Was he fleshing out things that hadn't even been implemented yet? Or was he describing--

Gosling: There were a few things. I mean, mostly, you know, there were places where there were vagaries where things had been kind of glossed over. There were things that in hindsight were just bugs. Sometimes he fixed them, sometimes somebody else fixed them after Guy pointed them out. You know, there was all kinds of waffle-ishness in the spec all over the place. And Guy really sort of tightened all the nuts and bolts, and he just did an awesome job.

 © 2019 Computer History Museum

Hsu: Before lunch we mentioned Jim Mitchell. I know this is maybe skipping ahead a bit, but can you talk about working with him?

Gosling: Loved working with him. He turned out to have quite a knack for politics that I sure didn't have.

Hsu: And his role was Head of JavaSoft?

Gosling: Well, no he was never that. He was certainly Head of Sun Labs.

Hsu: Sun Labs, right.

Gosling: But you know, he got involved in some of the standardization efforts. He got involved in a bunch of the politics with a bunch of the competitors. One of the things that sort of happened in that era, that was sort of weird was that in around '93, I started developing really bad carpal tunnel syndrome. And by the time that '96 rolled around, my hands were essentially just clubs made of meat I could have-- you could have pounded a nail through my right hand, I wouldn't have been able to tell. But I was doing all these different treatments. And carpal tunnel is a funny thing, and I didn't get the surgery done until around '99 or so. So since I could hardly even-- I could hardly type emails, you know, so I ended up having to be a talking head, because that's most-- and because me and politics, it just doesn't work. So I kind of disappeared from things. And people like Jim and that, they really took over. Certainly he was doing a lot of the relationships with other companies. And yeah, I liked him a lot.

Hsu: I think we were talking earlier about sort of the feature set of Java. How were those sort of-- how were those decisions made? Were there regular meetings? Were they mostly made in Silicon Valley? Or were you flying over to Aspen to meet with Bill Joy?

Gosling: Well, that was sort of a function of time. For the longest time, the only one who made any decisions about it was me. And I sort of had this rule that I wouldn't even-- if there were other engineers who were using Java that I wouldn't do things just because somebody said, "Oh, that would be cool!" Until somebody was like beating my door down, and really made a case. And that's really the way things worked with Bill, too. And sometimes we'd have those discussions at Aspen, and sometimes we'd have them other places. Mostly they happened in California. But I have a strong value to simplicity, and I don't like complicating things unless they really need to be complicated, so it needs, you know, for me, it requires a pretty compelling case for complexity. And you know, often when people want to do something like that, they just sort of do whatever comes into their head, and it's kind of like the first solution wins, and you know, with some of these things that Bill was wanting, the path to something that would really be effective was *not* clear. And yeah, so we were, on a few things, we were on an impasse. But yeah, there was nothing like a formal process, and certainly Sun was not a company for processes.

Weber: This might be the right era, you didn't actually-- you said just a very brief thing about HotJava, the browser and then the server. Is there anything more to add about as a browser, the decisions that went into it? And did that increase the pressure about features, once it became--

 © 2019 Computer History Museum

Gosling: The browser didn't-- none of them really created, you know, neither HotJava, nor any of the server work really created a lot of like direct language features. They certainly pushed some of the APIs. So like during the HotJava period, I ended up working pretty heavily on the networking APIs. So you know, people who-- I get all the blame for the URL class. And things around that-- but the language itself, yeah, not much.

Weber: And so HotJava was written in Java, and it was just nothing particularly remarkable about it, except that it was a web browser.

Gosling: Well, it was a web browser, but it was also one where you could embed new code. So you could put animations and interactive things into web pages, and it would take quite a few years for that to start showing up via JavaScript, and various other things. But yeah, Sun sort of decided that the landscape for web browsers was a really, really crowded and it was kind of hard to know why one would be trying to compete with Microsoft and the total horror story that Netscape had become.

Weber: But I guess that-- I had just assumed it was done as a reference browser, so that's how it ended up, but there were thoughts earlier on of trying to be a general, okay.

Gosling: Yeah, yeah, there definitely were. But you know, that whole area kind of became something of a bonfire for everybody in the way that the guys at Netscape basically committed suicide, and in incredibly public form was <sighs>--

Weber: What? By not keeping up better with Microsoft?

Gosling: Well <sighs>, they really pissed off Microsoft. I mean, they went out of their way to piss off Microsoft. There was sort of a fake ancient Chinese proverb that sort of had some circulation, which was, "When building a cage around a sleeping tiger, don't poke it in the eye." And they just kept behaving kind of crazy. And then this bizarre thing happened where the-- as the HTML spec grew, it turned out that, well, there were various things about the inside of the Netscape browser that the way that it was built was just crazy. And so as the spec evolved, and in particular, once this thing called the DOM, the Document Object Model, really came out, there was no way, really to implement the DOM on top of the Netscape browser, it was just nuts. So they started this second project that eventually became Mozilla, because they just had to start from the ground up. But the Mozilla project just took too long and got too crazy, and they died before it was really finished.

Weber: And actually one question from Lou Montulli, of Netscape, employee, he asked about the copyright of the Java APIs, and I guess because you did copyright, and whether, about Java openness in general. What's your feeling?

Gosling: So we tried really hard to be very open. I mean, we were distributing the source on Day 1. It wasn't under any of the sort of Free Software Foundation endorsed licenses. We had this horrible tension. Because on the one hand, for the vast majority of people on the Internet, we really, really, really wanted to just open source it all. Do like a BSD style license, and life would be fine. The problem is that we had

 © 2019 Computer History Museum

half-a-dozen competitors that were always up to games. And you know, it depends on the day of the week. I mean, the most common offenders were Oracle and IBM. Some of the time HP was being really bizarre. And it was just a com-- and Microsoft was being really bizarre. Pretty much any largish company in that space was screwing around, normally in ways that most people never saw. But we sure saw! And the kind of weird elaborate dance that we were doing that made no sense to anybody was making sense to us, because of "the way that Oracle tried to screw us last Tuesday, or the way that IBM tried to screw us the week before." And you know, any of those guys would have crushed us in a heartbeat, and we were trying to not get crushed.

Hsu: How much of contributions from outside were you actually incorporating?

Gosling: Actually quite a lot. In the early days, IBM contributed an immense amount. Various outsiders like the concurrency packages came from folks like Doug, Doug Lee. It really was an immensely large community effort. You know, we came up with this thing that this Java Community Process, and that was largely Jim Mitchell's doing. And that was all about getting a way to get the community organized, and in particular to do it in a really transparent way, so that-- I'm trying to choose my words carefully here-- because what we were trying to do was both eliminate as much of the political game-playing as possible. And when it happened to try to make sure that it could only happen out in the open. You know, we were trying to make sure that nothing could get in that didn't-- that was like the result of a backroom deal. I mean, sometimes there were things that involved ungodly arm-twisting. So like one that cost me about a year of my life was one of our-- we had a QA team in Russia that was all like PhD mathematicians, and they were doing a lot of the numeric QA stuff. And they wrote the most insanely difficult test suites ever. And they discovered a couple of Intel chip bugs. And of course then it turns into, "Well, are those a bug-- are those bugs, or are they exploiting vague corners of the floating point spec?" You know, the IEEE-754 spec. And you know, it caused some really strange behavior. And so Intel, after much arm-twisting caused this keyword to appear called Strict FP, which was a complete no-op on every architecture except Intel. And it's actually a no-op now on all the Intel architectures, because they changed the way they do rounding. And they've added-- so their sine and cosine functions did range reduction incorrectly. And then in later generations of the chip there were extra sine and cosine functions that actually worked correctly. So you know, we came up with workarounds for the sine and cosine bugs, which caused sine and cosine to be a little slow. So if you benchmarked C against Java, they would look to be pretty much neck-and-neck, except for benchmarks that involve sine and cosine. And the shit-kicking that Intel did because they couldn't actually fix these bugs, because if they fixed these bugs, then people's program behavior would change. And so that just kind of like crashed and burned. But if, you know, one of the things we were trying to do is to make sure that we could have some way to avoid getting totally slaughtered. So I mean, after a while when it had-- you know, we finally changed the license to GPL, which actually didn't change much in terms of what people could do with the source.

Hsu: And that happened in 2006? '05/'06?

Gosling: Somewheres in there.

 © 2019 Computer History Museum

Hsu: Yeah, yeah. I guess this is a related thing, but I mean, very early on the strategy for rolling out Java was to make sure to just get it out there as widely as possible, rather than to try to generate revenue from it. You know? And I mean how-- that was-- I mean, was that Scott McNealy that helped make that decision?

Gosling: Yeah, Scott McNealy was certainly involved, but this wasn't an act of charity. And you know, if you remember-- well, I don't know, you probably weren't around then and involved in things-- but that was a period when Microsoft was really terrifying the whole industry. And if you were a computer supplier like Sun or IBM or HP or DEC, anybody like that, one of the things that had happened was that Microsoft hired the Windows NT team from-- or the VAX VMS team from DEC, who became the Windows NT core team, and they started building Windows NT. And so before that their OS core, MS-DOS was just laughably stupid. And all of a sudden it looked like, "They're going to have a real OS underneath. And they could start doing sort of higher scale stuff than just running VisiCalc, and it would be like a real OS you could write real software on. And one of the things that happened when that started, when NT started to roll out was if you looked at the community of companies that provide software, they were in a quandary, so if you were a computer company like IBM or Sun, you know, a computer is just a useless bucket of metal and silicon, unless there's software on it. Right? So as a hardware manufacturer, you're critically dependent on all the software vendors. And you know, you really care about things like how thick your catalog is of software. And what was starting to happen at high speed was that, you know, these software vendors really couldn't afford to do versions for all the different platforms out there. And so there was a community of companies that were largely UNIX-ish, but not completely. And Microsoft had always been kind of weird, and because of MS-DOS, just completely unusable. But then NT starts to happen, lots of enterprises start making positive words about it, then all these software vendors go, "You know what, you know, for our own health, we have to switch our development over to NT, because that's where the volume is going to be. And that was also creating another weird problem for the community of customers, because a lot of the software was-- you know, its scale was limited by the scale of the platform. So like if you were a company that made hotel reservation software, you could handle small hotels on Windows NT, but if you wanted to run a hotel chain, like a big one like Sheraton, it couldn't run on Windows NT. But it could run on Sun, but from the vendor's point of view, it's like, "Which do you go for? The small number of large customers or the large number of small customers?" And more often than not they voted for the large number of small customers. So the-- you know, everybody had these software catalogs, and the software catalogs were all collapsing. And since none of these companies really made a huge percentage of their revenue out of software and they were all being sort of killed off by Microsoft, who was essentially sucking the oxygen out of the air, it was somewhat opportunistic for people to go, "You know, if we can make all of these-- all of our different platforms look like one platform, and run on Windows, then-- and then get software vendors to run on that one platform, then all of a sudden this flight away from our hardware stops, because it doesn't matter," right? And folks at IBM and Sun and HP didn't really care whether it ran on Windows or not, what they really cared about is whether it didn't run on their platform. You know, so folks like IBM and Sun, they were interested in selling hardware to the Sheratons of the world. And if they could make it so that the software developers could-- you know, didn't have to develop like a whole lot of different things, then all of a sudden, the world worked better for them. So it was really about trying to level the playing field for software developers so that the software catalogs didn't shrink.

 © 2019 Computer History Museum

Hsu: Right, but the strategy was that-- I mean, because Sun could have pursued a model of, "Well, we can do that, but we can also try to make revenue from Java, but--"

Gosling: But the problem with that is if you try to make revenue from it directly, you slow it down.

Hsu: Yeah.

Gosling: Right? And the direct revenue that one would make off of a Java Runtime, or a Java Compiler is microscopic compared to the revenue we'd make from a large machine, you know, like an IBM mainframe or something. So the making Java free was chump change.

Hsu: So Java would help sell more Sun workstations.

Gosling: Well, more Sun servers and Sun everything, and it would help sell-- it would-- right.

Hsu: Yep.

Gosling: Right, so in some sense it was kind of a weird marketing campaign to free the rest of the industry from the stampede that Microsoft was creating.

Hsu: I want to talk about the timeline. So Java comes out in 1995, and then-- and you're still part-- like at what point did you join Sun Labs? And then at what point did sort of the whole effort spin off into JavaSoft? Like what sort of was that timeline?

Gosling: So it-- I think that it sort of became a part of Sun Labs in like early '94, and it was sometime in like '95 or '96 that JavaSoft was formed. But like I said before, Sun wasn't exactly a place for org charts.

Hsu: Okay. <laughs>

Gosling: So you know, it was a pretty fluid place.

Hsu: Right. So JavaSoft, at the point where JavaSoft was formed, it was clear that it was becoming a big enough of a business that they needed an independent unit?

Gosling: Yeah.

Hsu: Mm hm. So at some point, Jonathan Schwartz became the-- joined JavaSoft?

Gosling: Um, um, um, um, um.

Hsu: So he had, his company, Lighthouse Design, had been acquired.

Gosling: Right, they got acquired. That kind of fell apart.

 © 2019 Computer History Museum

Hsu: And they were NeXTSTEP developer before.

Gosling: Yeah. And nobody understood why we bought Lighthouse. Everybody that I know that was asked for a vote said no.

Hsu: Sun, also, had a license to OPENSTEP at the time, too, right like in ‘95?

Gosling: Yeah. And by then I had nothing-- I was away from all that stuff. So I don't know what that license was like. And as near as I could tell the people at Sun who were driving a bunch of that were just crazy.

Hsu: So that was sort of happening completely independently from anything that you were doing or Java was doing.

Gosling: Yeah, I wasn't involved in that at all.

Weber: Were you involved with W3C standards very much?

Gosling: A little bit. In ’95ish there was this thing called HTTP-NG. So the basic HTTP protocol spec was a fine thing for an afternoon hack but it had gotten really way beyond its original use case and needed some serious work. And this guy, whose name I forgot, wrote up a proposal for something that came to be known as HTTP-NG. I really liked it. So I started getting involved in that and I actually did an implementation of that. Because I think his proposal originally came out in ‘94. And I thought wow this makes a bunch of things way easier with the way that we would do applets and things. Then something weird happened and I never quite understood what it was. It was like he had a health problem or something and he kind of disappeared. Nobody really wanted to move forward with it because it was kind of his baby.

Weber: But in general, though, Sun and JavaSoft were significant members of W3C?

Gosling: Yeah. But the parts that were around HTTP and that we didn't get terribly deep with. Yeah. But for myself the only thing I really got into in any depth was the HTTP-NG stuff.

Hsu: I want to talk about sort of the libraries. So there was AWT and then that was replaced with Swing, another graphics library. So why did that have to be replaced?

Gosling: Well, because AWT, quite literally what happened was we had this deal with Netscape where we would incorporate the browser into the Netscape browser. We would incorporate Java into the Netscape browser. And at one point, Netscape told us “You have six weeks,” and we were just getting started on doing and figuring out with the right UI toolkit was because the stuff that we had had before was clearly inappropriate. So we had this panic and it's like six weeks? You guys are nuts. You have no chance of being ready because we knew kind of what was going on inside because their offices were just a few blocks from where we worked. And so we just did the fastest, easiest thing we could do that we

 © 2019 Computer History Museum

could get to be vaguely runnable in six weeks. So that was a bunch of wrappers over the underlying UI toolkit. And we had to wrap both Windows and the UNIX Motif toolkit and those were pretty different. And so to build wrappers that would encompass both of those at all well-- it was fairly ungodly. But it got us out of all the hard bits of trying to do rendering and trying to do some of the tight plumbing into the UI system of the OS, all of which were really, really hard to do. Some of the things that were just too ugly to do in six weeks we left out, like really decent support for cut and paste. So the thing that was AWT, it was something that we did in a mad dash and it just never felt right. It worked just fine. It had a number of issues but because it was trying to run on all of these different toolkits it became kind of a lowest common denominator thing which was sort of okay for a while. But then—and even though IBM had participated in making it happen they got really militant about how it just was not going to scale. They couldn't add features to it because it was dependent on the underlying OS. And we knew it was a gross and ugly hack but it's what we had to do to make Netscape's deadline. So then we started the Swing project and we figured we could do it in six months instead of six weeks. So it was somewhat more thoughtfully engineered. But some of the things ended up kind of leaking through because we needed to interoperate so there are things in the, like the event model that kind of leaked through. It was sort of comical when some years later when IBM released Eclipse, it had yet another windowing toolkit in it that was almost exactly a clone of AWT which they had so soundly condemned but their-- what did they call it? GWT? It was like they did a wrapper that only covered Windows. So they only had to do Windows so that made it much easier but it wouldn't run anywhere else. And then they started this ungodly effort to make it work in other places. So eventually they got GWT to run in another places but it was just so comical that it was essentially a clone of this thing that they had been so angry about. So, you know, politics, what can I say?

Hsu: Yeah. I had heard that there had been a group inside Sun that had wanted to do a GUI framework that was more like NeXTSTEP.

Gosling: Yeah. And really what was-- the piece that was missing, I mean in most ways Swing was like the NeXT framework. The thing that was different was that in NeXT they had these, I forget what they were called, they were these files that were essentially a pickled-- or a persistent file that represented a UI. And so that in order to build a UI instead of saying…

Hsu: Oh, the nib file.

Gosling: Yeah, the nib file. So that you would just say "read the nib file" and it would create it.

Hsu: They're like freeze-dried objects.

Gosling: Yeah. So if you look through the Swing spec one of the pieces of careful wording in all of them is about the-- if you try to persist an object of a button or whatever that would give you-- it gives you the sort of information about some of the funny corner cases. And when that was all built originally the intent was that that would be a nib file. It would relate to the Swing equivalent of a nib file. So one of the things that's kind of awkward about Scott [McNealy], oh, it depends on which side of the fence you're on, Scott really didn't like competing with customers. So there were a number of companies that were doing developer tools, people like Borland and Symantec. And so he didn't want us competing with them. So we

 © 2019 Computer History Museum

actually never built the tool that would do the nib file thing even though we had built a bunch of the infrastructure for a nib file equivalent. And, of course, once Microsoft hugely dropped the price on Visual Studio and then all of a sudden companies like Borland and Symantec, they couldn't charge enough for their tools to survive. So they all just kind of like died. And then we ended up buying a company called Forte who didn't really do useful tools. I have no idea why we bought Forte, but we also bought NetBeans. And NetBeans was wonderful. But by that time the sort of standard practice had evolved to where people were building UIs just by writing code that said “new button this,” “new button that,” add them to that container, da-da-da. And even though it was not as elegant as the sort of pickled nib files it's what people were doing. So the NetBeans UI tools, they generated code rather than nib files. And although they did actually have-- if you dig down in the NetBeans metadata files there's something that's a lot like a nib file, you just never see it. And when your code gets produced, the NetBeans kind of equivalent of a nib file turns into a piece of source code and turns into a .java file.

Hsu: Can you talk about splitting Java into the Standard, Enterprise and Micro editions in 1997?

Gosling: Yeah. Well, that's because the business was going in different ways or there were kind of like really different businesses. And what was different about them was not so much-- it didn't really have anything to do with the language itself but it was all about the libraries. Right? So the Micro edition was all about cellphones and you certainly didn't need a fancy UI toolkit. And, especially, for cellphones of that era, they were too tiny to have a fancy graphics library. And then on the other hand the enterprise folks they wanted all of this complex stuff, everything necessary to run a large terabyte kind of app server with links into databases and all of that. And really it was just libraries. But for kind of marketing reasons they decided to call it editions of Java. And, you know, whatever.

<group laughter>

Hsu: One thing that I think was really important for the long-term adoption of Java was how quickly it was taken up in the universities in undergraduate curricula.

Gosling: Yeah. And there, one of the things that was interesting was that if you look at it sideways it's a lot like Pascal in the sense that like in Pascal if you have an array subscript out of bounds you get told immediately. Null pointers do ugly things. You get yelled at. Pascal is always considered a much more friendly language to teach people because it failed in clearer ways, kind of the rules made a little more sense. Whereas, if you're writing C programs there are all kinds of mysterious ways that you can screw up and nothing actually bad happens until long after the real error. And teaching kids to program C is mostly an exercise in frustration. And so Java comes along and it's got a lot of the sort of ease of learning that Pascal had plus it was something that you could actually turn into a career. And one of the problems with Pascal was that it never had any kind of commercial adoption. There were a few places people who were using some of the Borland tools but that never got a lot of adoption. And kind of the Borland experience with Turbo Pascal was one of the things that sort of fed into the discussion about should we give Java away? Because Turbo Pascal was pretty cool but it had a price tag that many people found kind of daunting. And it's not so much would big enterprises find it daunting? But would somebody in a back room find it daunting to just try it out. Most of the adoption [of Java] came not from people at the top

 © 2019 Computer History Museum

saying, “We will buy this”. It came from people at the bottom who were screwing around. They try an experiment and go, “Wow, this is cool”. Or, “Wow, I just tried re-implementing that transaction server and there are no memory leaks. It just keeps on running day after day after day.” You know? And then the results of those experiments percolate up and people start going, “yeah.”

Hsu: And I feel like, you know, I think Java is, in some ways, maybe responsible for sort of the industry-wide movement into object-oriented programming partly due to the fact that it was being taught in colleges now.

Gosling: And high schools.

Hsu: Because I feel like before Java there was sort of a reluctance. You didn't necessarily want to teach C++. There was a reluctance to teach-- it seemed like Java provided object orientation but also something that was used in industry. And there's all these things going for it that made it a good language to teach in.

Gosling: Well, and it was simpler and cleaner. Things like multiple inheritance in C++ often really confused people. And templates were always a source of much confusion just because they’re a source of much abuse. The way that people use the C++ templates is--it just begs to be the winner of the obscure programming contest. And some of that comes from C++'s heritage as a macro preprocessor on top of C. So it [Java] was cleaner. It was kind of weird. But certainly Java it did help make object-oriented programming mainstream kind of the way it made multithreading mainstream and garbage collection mainstream.

Hsu: In 2000 the original JVM was replaced with HotSpot?

Gosling: Was it 2000? Yeah, I think that's about right.

Hsu: And HotSpot had originally been developed for Self which was a variant of Smalltalk? Or grew out of Smalltalk?

Gosling: Well, it was a bunch of people who really liked Smalltalk but it was-- I never really got into the whole gestalt of Self very well. But it was these people who were trying to do a little start up around some hot compiler technology to try to make Smalltalk and Self fast. There really wasn't a market for Self because essentially nobody used it. But they figured that there was a market for Smalltalk. But the market for Smalltalk was pretty microscopic. And so Sun bought the company but it was mostly a bunch of folks from Stanford anyway so that worked pretty well.

Hsu: So what advantages did that new VM have over the old one?

Gosling: It had a really good compiler. And the previous one, it was just an interpreter. There had been, for a little while, there had been the start of a compiler.

 © 2019 Computer History Museum

Hsu: You mean a just-in-time compiler?

Gosling: Yeah. But that one kind of died because the motivating performance problem turned out to have been because of a bug. And so there was one big fixed in HotJava that all of a sudden made HotJava go like a rocket. And then it's like this just-in-time compiler that was-- it was starting to turn over and it was well past Hello World but it got shelved because there was just too much other stuff to do. And then the C compiler group built one that didn't really go anywhere because the only way that they could repurpose the C compiler to running Java was to break the Java spec which didn't go over very well. And then the Self crew that we bought, it took them a little while after we bought it to get something that really worked. So it does a bunch of moderately magical things.

Hsu: And the JVM has really become a host for a whole bunch of different languages from Scala, Groovy, Clojure, Kotlin, just to name four. It's become-- would you say-- well, I don't know if it's really eclipsing Java but how exciting is the work that is being done in the JVM?

Gosling: It's really, really cool. From my point to view all of the interesting parts of Java are in the JVM and many things about the language that kind of bugged me are all because of its history from C, some of which are slowly getting updated, things like switch statements. They feel very seventies. But that's all sort of from my point of you the language syntax is kind of like window dressing; what actually matters is the engine underneath. The JVM has proved to be significantly more flexible than I could've expected. And for people that are experimenting with programming languages not having to build a crazy hot optimizer or a really high scale garbage collector is a big deal.

Hsu: Talk about how important Java was in fueling the dot-com boom?

Gosling: I honestly don't know. It certainly made building a lot of these giant web apps hugely easier. And once the people started doing app servers that could scale it certainly helped that a lot. Lots of people did formal evaluations of the programmer productivity, like Java versus C and all the studies were better than 2X. And so that certainly let people build a lot of stuff in the dot-com era. But really the dot-com era was more about exploring weird and wonderful sort of business models and some of them worked and some of them were just crazy. And it fundamentally ended up as like this big filter function.

Hsu: So at some point Java, you sort of stepped back from day-to-day work on Java and went back to working in Sun Labs just doing research. So what were you working on at that point?

Gosling: I mostly stepped back when my carpal tunnel got bad. And then after I had the surgery four years later I could actually build stuff again. But by that time the whole process had become so political the way that this community process with the sort of half-dozen kind of bullies was really difficult for me because I had been involved in some of that when kind of all I could do was talk but I wanted to do something that felt a little more productive. And once I got my hands back then I went into Sun Labs and just started building stuff. And some of it was just internal tools. One of the bigger things I did was I got involved in the NetBeans folks after we had acquired them and I did a bunch of prototyping of refactoring tools and that was just huge fun.

 © 2019 Computer History Museum

Hsu: We could skip this if you want but what's your view of Android's use of Java?

Gosling: It's a mixed bag. The fact that they use it I think is great. It helped them a lot. The way that they and particularly Andy Rubin treated Sun, another story.

Weber: Is that one you're willing to tell?

Gosling: Well, if you want details you should talk to a guy named Vineet Gupta [ph?] and he will tell you in gory details. But fundamentally what he wanted us to do was to do a total open source in a way that would give them all the stuff for free. So the license that we had was allowing us to get revenue from companies like IBM as a part of maintenance contracts and stuff like that. And really the thing that we cared about the most that was in the not exactly open source agreement was about maintaining interoperability. And we really cared about interoperability. I mean that was one of the big-- but everybody-- I mean it's kind of like all standards. The people who join standards associations is to promote the standard. They all say they want to promote the standard but really what they want to do is turn the standard into an entrapment device. So they'll implement SQL but they'll add extensions that make it impossible for people to run their apps on other platforms and do it in a way that they'll add extensions but then there'll be patents on the extensions so that other people can't. So it's kind of the worst sort of forking that kind of everybody wanted to do. And they wanted an arrangement where we lost all possibility of any kind of revenue and all possibility of any kind of enforcing interoperability of any kind. And we didn't actually care about the revenue as much as the interoperability thing. And we made them some proposals that would've been pretty cheap and, I think, in the end would've been fine for them because they eventually realized that the universe has a lot of tooling around Java and if you start breaking things by too much things fall apart. And one of the things that the Android folks had done was they had adopted this graphics library that was kind of crazy but they adopted it because it was free. So we actually made an offer to them that was actually pretty cheap. For a company like Google I'm sure they paid more for that for lunch in a day. But they didn't want to be part of a community of people that sort of were in the debate. They just wanted ownership. And it would've been so much cheaper for them to do that than engage in this crazy lawsuit with Oracle. So, like I said, it's sort of a double-edged sword. It's nice that they did that. But how they sort of raised the pirate flag and the middle finger and were actually pretty abusive to a bunch of people was not good. So I often get asked about the Oracle-Google lawsuit, which side is right and it's like neither of them are right. They're both bad actors. Although in Google's defense it was-- a lot of the misbehavior was centered on one individual. But since he was the guy in charge it was like…

Hsu: Are you comfortable talking about your six months there at Google in 2011?

Gosling: It was odd. I've only got a few minutes left but there was-- I felt a certain sense of bait and switch because it became really clear that one of the reasons that they wanted me was for help in their court case with Oracle and I thought that both sides were pretty reprehensible. And when they were trying to hire me I said, look I can't participate in that court case. They said, "Yeah, fine". And then afterwards lawyers started calling me up saying, "Well, so we'd like to get your opinion on…" Like, sorry. And I was asked, "Can you participate, can you testify to this?" And I said well I could testify but I guarantee you

 © 2019 Computer History Museum

won't like my answer. There are no answers I can give that will make you happy or improve your case. And just like the crazy ageism at the place. That's very real. There's a certain amount of that at most of the companies in Silicon Valley but Google certainly six years ago was, seven years ago, eight years ago? was pretty extreme. And I have a number friends who were there for quite some time early on and the gray hairs are a huge problem.

Hsu: To close out, could you talk a little about your work at Liquid Robotics and now at Amazon?

Gosling: So Liquid Robotics was probably the coolest most fun job I'll ever have. Autonomous robots in the ocean, having to commute to Hawaii, having an engineering test site that was on a wharf on Kawaihae Harbor. Most of my testing was done two or three miles offshore and snorkeling. So I can't imagine another software job that would require me to spend a couple of hours a day in the water snorkeling surrounded by ono and ahi and humpback whales. It's like oh, swimming with turtles and it's a job. Plus, you know, doing software for collision avoidance of these robots that have to be out there for months and months and months at a time and surviving in extreme conditions, that was just huge fun. And the devices were really, really cool. But then when it turns into hunting submarines for Trump after getting acquired by Boeing, yeah, it wasn't working for me. Now, I'm at Amazon which is a company that I'm really enjoying. I've been there for a couple of years. I like the fact that there's no inherent conflict of interest in their business practices. At places like Facebook they're fundamentally profiting off of your personal private information. Their revenue is a violation of your privacy and Amazon's revenue, you know exactly what you're paying for with Amazon. There's no hidden agenda. Things are pretty obvious. And all ages of people, all nationalities, it's a real blend of people. It's an enormous company but, you know, pretty good.

Hsu: And you work there is also Java-related, right?

Gosling: It's kind of all over the map. Yeah. There's some of it that is quite Java-related.

Weber: What was your one word for young entrepreneurs?

Gosling: Play.

Hsu: All right.

Gosling: All right.

Hsu: Thank you very much.

END OF THE INTERVIEW

 © 2019 Computer History Museum