Oral History of James Gosling, part 1 of 2

Interviewed by:
Hansen Hsu
Marc Weber

Recorded March 15, 2019
Mountain View, CA

CHM Reference number: X8971.2019

Hansen Hsu: Today is March 15, 2019. I am Hansen Hsu.

Marc Weber: And I'm Marc Weber.

Hsu: And we're here with James Gosling. So you grew up in Calgary, Alberta?

James Gosling: Yes. I grew up in Calgary Alberta.

Hsu: And you went to college there?

Gosling: Yes. I went to the University of Calgary.

Hsu: And you studied computer science?

Gosling: Yes. I was a computer science major, actually, kind of an over major because I actually technically didn't complete the requirements to graduate. I took too many computer science courses.

Hsu: Was that what you wanted to be when you grew up?

Gosling: I never really thought much about what I wanted to be when I grew up. That wasn't part of my thinking. It was more, hey, this is fun. I like having fun. I want to do more of this.

Hsu: And you went on to graduate school at Carnegie Mellon?

Gosling: I did. I went to Carnegie Mellon.

Hsu: And you did a lot of work on UNIX while you there?

Gosling: Yeah. I spent a lot of time doing work on UNIX at Carnegie Mellon. One of my favorite projects there was doing a multiprocessor version of UNIX.

Hsu: One of the things you're known for is an early version of Emacs on UNIX.

Gosling: Yeah. I did the first version of Emacs in UNIX. I had been exposed to Emacs on Multics. And there was also an early version that existed on PDP-10s. And when I did this project to do a Pascal compiler on Multics I used Emacs there and then I sort of get back to grad school and it's like, I miss Emacs. So I needed to build one.

Hsu: You first started to work with virtual machines and byte code at Carnegie Mellon?

Gosling: Yeah. So I got started doing work with byte codes at Carnegie Mellon. It was a project where my thesis advisor, a guy named Raj Reddy, had asked me to port some software from these PERQ

computers from the Three Rivers Computer Corporation, port them to VAX and I'm not sure what he had in mind. But I decided that, you know, porting all of this code would be really hard. But building a byte code to VAX assembly code compiler would be really easy and it was. And the thing that astonished me was that the code quality that I was getting was better than the native C-compiler and that worked really well. And I just sort of kept that little nugget knowledge hidden away for a few years.

Hsu: Tell us how you got to Sun.

Gosling: I got to Sun through a somewhat odd course. I had been-- because at Carnegie Mellon I had got to know pretty much everybody in the BSD universe because it was a pretty small universe so I got to know people like Bill Joy and Kirk McKusick and Eric Allman and Andy Bechtolsheim who, oddly enough, I had gotten to know him somewhat because he had been a student at Carnegie Mellon before he went to Stanford. And through an odd twist of fate I actually had lunch with Andy at about the time that he was signing the papers with the VCs. And I thought this whole start-up thing with these wimpy 68000 processors was kind of lame. And at the same time, I was interviewing at IBM and IBM had this really fabulous microprocessor. So I went to work for IBM. And that turned into a year-and-a-half long teaching moment about how cool tech never wins out over stupid bureaucracy. And then-- the whole time Bill Joy was like actively twisting my arm and saying, "Come join Sun. Come join Sun." And eventually I sort of gave in.

Hsu: And one of your early projects at Sun was NeWS, a remote windowing system?

Gosling: Yes. It was really my first project was the Network extensible Window System which was one that let you put windows up on remote computers. And the protocol was literally PostScript.

Hsu: It was like X Windows crossed with Display PostScript kind of?

Gosling: Yeah. I had done a window system before this. It was one of my central projects when I was at IBM. And then there was this project at MIT that was a part of Athena that also did a distributed window system. And that was a thing called X10 and it had a large pile of problems. And then I came to Sun and I started working on NeWS. And I had learned a bunch from the previous window system that I had done when I worked at IBM so I tried to clean up a lot of that. And then the MIT folks they then sort of re-architected and built X11.

Hsu: So the two projects were actually competing with each other?

Gosling: Yeah. I mean effectively. The NeWS project started before X11. And then the MIT folks started X11. And eventually we tried to do a merge-- which was their models were really different so it didn't-- the whole merge thing was kind of weird.

Hsu: So then a couple of years later a group of you including Patrick Naughton were unhappy about Sun missing out on the potential for consumer electronics?

Gosling: Yeah. So there was a group of us, the core group was about four people who were kind of cranky. Well, it's sort of a combination of cranky and fascinated by the fact that there were things going on with computers that were outside of the normal computer industry. You know, people in industries from early telephone handsets to people doing elevators and locomotives and lighting control systems and all of that. They were putting computers in them. And we thought this was fascinating. And it felt like there was something that we were missing out on. And one of the things that, like the folks at Apple were doing and then Steve Jobs when he went to the NeXT, was saying consumers are relevant. And we thought, yeah. Right? And this whole big community of engineers who were building things that are outside of the computer industry, that was pretty interesting. So that started us on this exploration of issues.

Weber: Was there an important sort of aha moment in a hot tub, I think, about that?

Gosling: Oh, that. Yeah, that came somewhat later.

Weber: Okay. We'll get to it when…

Gosling: Yeah.

Hsu: A number of you were-- at least Patrick was threatening to leave Sun and join NeXT, is that correct?

Gosling: Yeah. He was particularly cranky and was thinking of going to NeXT.

Hsu: So that was the beginning of the Green project?

Gosling: Yeah. Yeah. So, and there were a bunch of us who had been talking and Patrick got particularly cranky about it. And so Scott said…

Hsu: Scott McNealy?

Gosling: Scott McNealy said, "Why don't you guys go off and think about it? And I'll fund you for a year or two and tell us what you figure out."

Hsu: So that led to the creation of the Star7?

Gosling: Yeah. Well, there was kind of an intermediate point there where-- so we got this Green project started and one of the people who was a part of that was a guy named Mike Sheridan. And he's got an interesting background but he's not an engineer. He is much more of an artist/MBA business kind of guy. And he held the pen but we all sort of worked on this document that was titled "Behind the Green Door". And we literally named the project the Green project because we had rented some space on Sand Hill Road, that the door into the space from the elevator lobby was green. And we're not particularly creative when it comes to things like that but we take advantage of opportunities when they come by. Yeah. I think the number one reason we picked that office space was because the door was green. But anyway we kept working on this document that was kind of a collection of scenarios and ideas and such. It was kind

of a business plan. And it was informed by a bunch of road trips we went on. We went and we visited a lot of companies that were kind of outside of the computer industry but using computer parts. We went to companies like Mitsubishi and Matsushita and Toshiba and Sony in Japan. We went and visited some folks at Samsung in Korea. We went all over Europe. We visited lots of folks to try to understand what they were doing. And that bunch of sort of ideas we argued over and that became the business plan for the Green project. And as we came out of that-- we were mostly a bunch of engineers. And, you know, particularly me, I'm not a big person for prose. And we decided that what we wanted to do to sort of explore the details of some of these ideas was to build a prototype, build a demo object that kind of encompassed some of these ideas. And, you know, it wasn't something that we were building because we thought it would be a product or anything. It was a vehicle for exploring ideas and that was this device that came to be called the Star7.

Hsu: And the Star7, could you explain what capabilities the device had?

Gosling: The Star7 was this rather odd ungainly device. Because we weren't trying to build a product we didn't do any really extensive engineering to come up with fancy parts. We engaged in what we referred to as "hammer engineering." And what we meant by hammer engineering was you take something else like a Sharp LCD television set, you hammer the TV to break it apart. You take the LCD screen out and you use that. We bought all kinds of weird consumer electronics gear, shredded it and took parts out of it. I don't know why but some of the electrical engineers fell in love with the connectors on Sony Walkmen so we bought a bunch of Sony Walkmen and ripped them apart to get the connectors. They had actually looked into buying the connectors and it turned out it was cheaper to buy the whole Walkman and take it apart than it was to buy the connector. So we did a lot of that. There were radios that came out of some military gear. There were batteries that were pretty off-the-shelf. There were PCMCIA cards that were early prototypes. There was a company that had an experimental technology for doing flexible boards that kind of worked. And so we figured we needed to use that just because it was weird. And we had these infrared transceivers that came out of something, I forget what. So, yeah, the thing was kind of a Frankenstein's monster wrapped in a piece of machined aluminum. So it was kind of bulky and odd-shaped but it had the pieces that we thought were interesting. And so it was vaguely a handheld machine. In our mind, and this was in 1991, if we ever turned that particular part of it into a product it would be kind of like an iPad. And, in fact, a couple of the people on the team went on to work on the iPad.

Hsu: But it was supposed to be a remote control?

Gosling: Well, it was supposed to be a-- so it wasn't so much that particular device that we were trying to do. That was just an instance of things which fit together in network devices. And, you know, the vision was all about building devices into your environment that you could—that could communicate with each other and that could be controlled remotely. One of our scenarios that we were trying to solve was the fact that TVs and VCRs of the day, they all had their separate remote controls and this felt like madness. And it was madness then. It's still madness. It's kind of crazy the lack of progress in that. I've still got this pile of remote controls on a side table at home. Actually, I've got a couple side tables covered in remote controls and it's just crazy. And it's been like that for 30 years. But, also, to be able to use the devices for interacting, to be able to control not just the TVs and that but the lighting and be able to interact with

friends, to be able to collaboratively watch movies, share all kinds of information. And there was also a big component that was the sort of software that lived on the devices because in order to become a full-fledged participant in the network things like TVs and amplifiers and that they all had to somehow or other grow. And so the Star7 was the central device that we built but it was just kind of a demo in the midst of the plan. And it's kind of weird these days to see 25, 30 years later that people are actually building that.

Hsu: How did you come to create a new programming language for the project?

Gosling: So we had a lot of interactions with the engineering teams at these consumer electronics companies and with the business side of their teams, as well. And we were trying to understand, you know, what problems they had. One of the things that struck me was the extent to which these folks were repeating the mistakes that the computer industry had gone through decades before. So they were doing things like stumbling their way into reinventing networking but making kind of obvious and crazy mistakes; things that were known to be bad ideas in the sixties. And even though I was still in my thirties at the time I had been through a bunch of that stuff. And it was really clear which crater this stuff was going to land in. So we were trying to bring together a bunch of different issues, sort of learn from them and help them learn from us. And it was never really about creating a programming language. It started out as the sort of realization that a number of the problems that they were having came out-- came about from some of the programming tools that they were using. And it sort of turned out as, well, are there things we could do to C and C++ to make them work better in this environment? And so I kind of volunteered as the only person in the group who had built a compiler before to sort of make C safe for that environment. But then as time passed it was like, well, this thing changed and that thing changed and the other thing changed. And pretty soon it didn't really look quite as much like C++ as I had imagined. And so it kind of took on a life of its own.

Hsu: And sort of what other languages were big influences on Java?

Gosling: Oh, piles of them. It sort of grew out of the C/C++ tradition but there was a lot of influence from languages like Simula and Pascal and Mesa. If you look sideways you can see influences from languages like Lisp and Smalltalk. So there's a lot of bits of many, many, many things in there.

Hsu: And initially the language was named Oak?

Gosling: Yes. Yes. It was initially named Oak. It was another one of those sort of lazy things. I had just been-- I had just figured out that I had to do this. And so I was sitting at my desk looking out the window, there was an oak tree. And I was literally typing “mkdir” and I needed a name for the directory to start putting stuff; that was outside my window. I just went with it.

Hsu: From the beginning the language was designed to be platform-independent.

Gosling: Yes. So platform independence was one of the things that started out at the very beginning. And strangely enough that actually came originally from the consumer electronics companies. And, actually, more from a bunch of conversations we had had with their management and purchasing people.

They had this problem where when you look at a device it's got lots of components. It's got LCD screens. It's got resistors and capacitors and boards and wires and connectors. And all of those are things that you can buy from any number of suppliers. And they really liked not being tied to any particular supplier because if one of the suppliers gives them a hard time then they can just change suppliers. But CPU chips were a different thing. And because there was so much kind of wired into C and C++ that let the underlying CPU chip leak through and the way that binary artifacts were created it meant that they had to make a decision about what CPU chip they were going to use at the very beginning of the project. And they were pretty much stuck with it forever because it becomes then very, very expensive to change. And so they found this to be completely unacceptable verging on terrifying because it meant that one of their suppliers had a noose around their neck. And so then the question became, well, what would you need to do to make it so that they could flip from one chip supplier to another really easily? And, you know, all of the stuff about like the way arithmetic is specified to the-- well, the pointer model came partially from that. But even things like byte codes came largely from that because go back to that other project when I was at Carnegie Mellon when I was porting these byte codes I had sort of went, these byte codes have some interesting properties and there were some problems here. But if you were to do them just slightly different how could you make it so that you could target different CPU instruction sets easily? So that had been set up in the back of my mind years before. And then when we started talking to folks in the consumer electronics industry then all these sort of supplier tie-in concerns came out. But there was also – that's like concerns as you go across the different vendors. But there was also an evolutionary problem. If you go along with the time axis instead that they would build something for some chip and then they'd go to do like the next VCR or the next television. And they would have to use the next version of that chip. And often they would end up having to rewrite huge gobs of stuff because the processor architectures had just evolved. And so one of the things that was kind of an important principle at the time which has actually worked out really well was to be able to take software and run it on machines that have not been invented yet. And people in the Java world do that all the time now.

Hsu: So Java is famous for the slogan "write once, run anywhere".

Gosling: Yeah.

Hsu: That's a direct result of this work.

Gosling: Yeah. And there are various different rephrasings of it which is things like "debug once, fix everywhere", "train once, work anywhere".

Hsu: Was that a marketing term that came out later?

Gosling: Yeah. Very much so. I have no clue who came up with that little line.

Hsu: It's a great line.

Gosling: Well, yes, it's a great line but it oversimplifies things. And me being a rather particular engineer I go, nyah.

Hsu: Talk about forming the First Person subsidiary.

Gosling: So the Green project started kind of at the very end of 1990. It went on through ‘91. And it ended in September of 1992. When we had built the Star7 we did a bunch of demos. We said look at this toy we built. Look at all the stuff we learned. And then at the end of that exercise we kind of went, there are some parts of this that were actually pretty damn interesting. And the Green project was really entirely about research. It was a learning project. So, you know, we got to September and the question then was are there things that we could do with this to turn it into a business, to turn it into something that, you know, could help us pay our mortgages? And so that's when First Person was created was an attempt to figure out how to turn this into something that was not a toy.

Hsu: And where did the name come from?

Gosling: I'm the wrong person to ask. I think that was a Mike Sheridan thing. It was all about trying to make the user be directly in the middle. So it was all about how do I interact with my world?

Hsu: And the project-- the product that First Person was working on was going to be like an interactive TV set-top box?

Gosling: Well, when First Person started we didn't know what we were going to do. We were just searching for commercial avenues that might make sense. And we had all come from a background that was Sun workstations and the Internet. Sun was the first company that really, you know, said that a core principle of everything is the network. And it attracted a lot of people who believed that. And, certainly, I believe that. And even looked backwards from today's viewpoint at some of the ugly things that have happened because of the network—you know, Facebook and the rest of it get really kind of offensive—but we were looking at ways to do something with this. And at about the time that we were thinking about this Time Warner, the cable company, there was a group of people at Time Warner that wrote up this proposal for something that they called the full-service network. And the full-service network was perfect. Their whole laundry list of what they wanted was exactly stuff that we had already built and believed in. So we started a long-- we just decided that that was it, build stuff for the Time Warner full-service network. And the full-service RFP when it got published, it made quite a splash in the industry. And so everybody decided that they wanted to do that, every phone company and cable company on the planet decided that they wanted in, too. And so we went, ooh, all of a sudden a lot of opportunity all lined up. And so we built prototypes, wrote proposals. We spent an ungodly amount of time working with the folks at Time Warner and mostly that was a learning exercise in the politics of these large media companies, which was mostly not a happy time. We came to understand the inner conflicts and people's understanding of their business models. And, in particular, the conflict between the people at these cable companies who thought that what they were selling was cable television. But in, actual fact, their business was selling eyeballs to advertisers because that's where the real money was but most people didn't really understand that. And, in particular, most people in the cable companies didn't understand that. So we had a lot of misleading and confusing interactions. And, you know, we sort of discovered that the “free and open, anybody can contribute content” [model] was directly antithetical to the “we must control the eyeballs” view of a lot of their senior management. So we spent a lot of time learning. But, also, the people at the cable companies

ended up learning because people with eyeballs don't want to be controlled. And people really loved this Internet thing that was happening and the cable and phone companies they hated the Internet with a passion that was really quite spectacular. But we eventually decided that the right thing to do was to just give up on the cable companies and the phone companies because they were just too conflicted and just repurpose everything for the Internet. And that was-- you had asked the question about the hot tub moment; that was the hot tub moment.

Hsu: Okay.

Weber: Do you want to describe that very briefly?

Gosling: Yeah. So we were coming to this realization that all of the work that we were doing with the phone companies and with the cable companies was going absolutely nowhere because in their heart of hearts they did not want this thing. They say that they want the thing but what they say and what they do are very different things. And, you know, we kept getting into these crazy situations. And because we were a bunch of engineers from Sun everything that we had built had been based on Internet technologies because it was the right answer. And Sun had this big offsite-- the company's leadership teams often get together. And this was at a ski resort in Lake Tahoe. And we weren't quite sure what to do. It's like jump off a cliff, get drunk and do something stupid. And we were-- it's this big ski resort. They have these giant hot tubs and some of us were there and just bemoaning how crazy this had gotten. And eventually the conversation turned to let's just do this where it works. Let's just stop the pain. And, you know, we had been-- the sort of reasoning that had been going on in our heads before that was that the phone companies and the cable companies were used to a business model that involved interacting with millions and millions of customers. The people who provided Internet service were used to research labs and universities and that sort of thing. They were never set up for scale. And a lot of things in the belly of the Internet specs were not really set up for scale. And so we had thought, if we really want to do a consumer play, cable companies and phone companies were the place to go. And then realizing that no these really aren't consumer companies, these are jellied eyeballs in glass jar companies. And we had been kind of watching kind of the Internet go through this slow transformation from research labs in corporations and universities to something that was much broader. And so we kind of had this epiphany that it was way more likely that the Internet was going to grow into something that would work with consumers, than that these consumer companies would ever become friendly with their consumers-- with their customers. And, you know, so we switched over and eventually the cable companies and the phone companies they kind of got reeducated just because that's what their customers want. And so that sort of switchover was a pretty traumatic moment. But it was definitely the transition between just one deal after another just dying crazily to, wow, things actually work here. And it wasn't a technology working. It was a business proposition working.

Hsu: And that was which year?

Gosling: That would've been '94 or '95-- well, '94 really.

Hsu: So the team then started working on a demo browser, what became the HotJava browser?

Gosling: Yeah. So we started working on a Web browser that we called HotJava. And the primary thing that we were trying to put into it was the ability to have live interactive content. If you think back to what a Web browser was in 1994 it was text, if you were lucky, a few images, and that was about it. There was no animation. If you clicked on a button it would take you to another webpage and that was it. There was no ability to do any kind of interactivity. And we had been building all this stuff that would let you build very interactive and dynamic experiences that could be exchanged across the network. And so we sort of built a browser that sort of encompassed that.

Hsu: Could you talk about the demo that you did at the TED conference?

Gosling: So at the TED conference it was either late ‘94 or early ‘95, I forget, exactly when-- that was another one of these sort of psychotic incidents where one of the characters that's a big part of this picture is a guy named John Gage. And John Gage is one of the best public speakers I know. He’s very passionate, very-- he's got a really good sense for society. But he’s, also, one of these people that’s on a mission. You can't actually ever tell John what to do. He just follows his mission, whatever his mission is. And so we had been working on all of this stuff. And because we’re a bunch of engineers we don't really think much about telling a story. But John, who was tied into the whole TED crew, he decided that he needed to do a demo at the upcoming TED conference in Monterey. And so this one day I'm sitting in my office and John goes down-- comes down the hall and he's pushing a cart where he's got some equipment on it. And as he comes by my desk he says, " James do you have a…" and I forget what he asked for. And I said, well, I don't know, I could probably find that. And I said, John what are you doing? And he said, "Well, there's this conference and I want to give a talk and I thought I'd grab some of your demos and sort of show them off." And I was like “but John nothing really works very well. It's not like you can just install it and run. And you need—" getting network conductivity is dicey. He said, "Yeah, but this is a really good place to show it all off. So I'm just going to go and show whatever I can." And, you know, this flashing red light went off behind my eyeballs that said “disaster, disaster” because the chances that John could actually get anything to work was pretty minimal. And since his car was literally illegally parked in front of the building while he scraped up all of the stuff, I said John I am coming with you. And so I got in the car with him with all of this stuff and we drove down to Monterey. We get there kind of late at night and we go to the conference center. I spent all night getting this shit to work, most of which was working with MCI to get their network configuration settled out because configuring Internet stuff was not something that the phone companies were really good at. And so when the time came for our demo we showed a lot of-- well, it starts off with John doing all of this talking about what you can do with the Internet and what you can do with documents and what you can do when things are more lively. And one of the applets that I had written was one that would display a 3D model of a molecule. And when we got to the demo part of it, you know, people were really used to seeing webpages that were just text and pictures. And so this webpage comes up that's text about whatever the molecule was and then a picture of the molecule. And I took the mouse and I moved it over to the molecule and I dragged it and the molecule rotated. And this was the first time that any of these people had seen an interactive dynamic object on a screen in a webpage like that. And the <gasps deeply> that happened in the audience was really amazing. And then at that it's like everybody was paying attention. And John kept spinning the story. We had a few other demos but it was really that rotate the molecule-- this thing that they thought was a picture just rotating. It was kind of like one of these, you know, in “Harry Potter” where the paintings

on the wall come to life. It was that kind of like whoa. And, of course, one of the parts of it that I took significant pleasure in was the folks at MCI screwed something up just before the end of our talk. And so the demo died while I was on stage but I was able to hand wave through it. But then the next talk which, I think was SGI, they got on stage and the network was dead because it was MCI. And I hope that nobody listening to this knows who MCI was, a long dead phone company. But, yeah, that was really the first public presentation that had a big impact. Of course, the fact that John Gage is like one of the best public speakers I've ever known had a lot to do with it.

Hsu: Could you talk about the deal with Netscape and how important that was?

Gosling: Yeah. So the deal with Netscape was really important in getting this embedded in browsers and spread around the world because they were the company with overwhelming market dominance in Web browsers. We did this deal with them to get Java incorporated into the Netscape browser so that that kind of dynamic content was available there. And that was kind of an insane period of engineering to try to get these code bases to all fit together. But in the end it worked reasonably well.

Weber: What was the larger vision for how Java would transform computing, potentially make operating systems irrelevant? That was the period when there was the most enthusiasm about Java running within every browser becoming really the future of the whole field.

Gosling: Yeah. And in retrospect there were a number of the key decisions that were made that came about from concerns from these consumer electronics companies. So the isolation from the containing platform turned out to be a property that was appealing in a lot of different domains, so the fact that it didn't really care what browser you were in. Now, admittedly, the process of integrating something like the Java VM into a Web browser was not exactly straightforward; and, in particular, on the Windows platform of that era it was kind of a nightmare. But the fact that sort of isolation of platform issues from the application had really huge implications. And as we started thinking about that not just from the point of view of sort of having come from this sort of consumer and embedded kind of focused mindset, to thinking about what this meant for a company like Sun was that, when you're a company like Sun what, that means is that you can take competitors and they can actually cooperate in a way that lets them tell a story to their customers that they are-- actually more important than their customers was the software development community. Because if you're a software developer and you build something like a piece of management software for a hotel chain, if you're developing for HP or Sun or Apollo or IBM and when you develop for those platforms you have to develop specifically for that so that your customer has to decide, has to make a hardware decision that Apollo is yours. Then, if you’re a company that's doing this hotel management software, you’re going to go for the highest volume platform you can get. And there was this conflict that lots of these folks had where the highest volume platform was becoming Windows NT. But the highest volume platform was not the most capable platform. So if you were a small hotel chain you wanted Windows NT. If you were a bigger hotel chain you wanted somebody like Sun. So what does the company doing the hotel management software do? So if we can tell the software developers a story that says if you develop on Java then you can sell to small hotel chains who want to run it on Windows NT and you can sell it to large hotel chains that want to run on a big Solaris box. And that way we kept the

software development community really engaged because when you're a hardware company a piece of computer hardware is just a brick unless there's software to run on it.

Weber: But were you hoping to actually take on operating systems and desktop applications?

Gosling: Yeah. Well, we were certainly wanting to take on desktop applications and we did. A lot of people did desktop applications in Java. They actually still do. But at various times we talked about being able to run Java straight on the metal. And we actually did some experiments with that. But it turned out to not be as valuable. So being able to run on top of UNIX or Windows or whatever-- because there was always something else that people wanted to run. And re-implementing all of the something elses was generally more than people were willing to bite off.

Weber: And how quickly did it become-- by '96 there was something like 100 companies on the Java side versus Microsoft, if I'm remembering correctly. It became a real battle.

Gosling: Yeah. And the Java side became sort of heavily dominated by people who were trying to work at scale. And I was in the wrong part of the business to really have a good view of why that all happened. But it's certainly the case that the Microsoft universe, since it grew out of little desktop units that ran Word and Excel, they were good for small things but they always failed miserably trying to do large things. And the rest of the computer industry had really been focused on doing large things. And so they took it where, you know, for them the money was. So it was companies like HP and IBM and Sun, they sort of took it up the scale, upscale.

Weber: And ActiveX, do you want to talk about that?

Hsu: Oh, yeah. I guess Java was really competing with ActiveX at the time.

Gosling: I think it was competing with ActiveX in the press but not so much technically. ActiveX had become an extraordinarily complex way of embedding components. And it had no chance of working anywhere other than on Windows. And, you know, when I talk to people I knew at Microsoft-- because the way that Microsoft is structured it's really a bunch of really independent groups who are all throwing bombs at each other. The ActiveX spec got really, really difficult. And, you know, because it had this complexity it didn't really solve people's problems. It tied you into one platform. From a technical point of view it was never really something that caused us a lot of pain. But from a marketing and PR point of view Microsoft had infinite dollars to spend on marketing and Sun never did.

Hsu: Speaking of marketing, I wanted to go back to talking about the name. So the Java name came up after it was discovered that Oak was taken by another company?

Gosling: Yes. So if you're going to release a product you pretty much have to name it something. And because lots of names are trademarked you pretty much have to trademark a name that you're going to use to make sure that you're not going to get sued by somebody else. And one of the first steps in that is you get trademark lawyers to do searches for possible conflict. And, you know, everybody just wanted to

keep using the word Oak because folks had gotten used to it. And the lawyers did their analysis and they found so many potential conflicts. It was kind of strange how many programming language software projects, the funny little things done by odd defense contractors or Lord knows what. And so there was no chance. The lawyers were just like, “No. You will die in a hailstorm of lawsuits.” So we were forced to pick a name. And, actually, it became kind of a common thing to have these arguments about, "Well, I like this name better. That name better." And there's this problem that if you try to pick a name that kind of relates to what you’re doing it's almost certainly already taken. And people would fall in love with this name or that name. It would be-- we’d ask the lawyers and they'd say no. And we actually got to a point where we were actually ready to release the software and the thing that was blocking us was a name. And so what we ended up doing was tracking down a person who was a product naming expert. We didn't actually engage them to do the product naming. What we got them to do was to facilitate a yelling and screaming session, which is the only real description for it. So what we ended up with after much-- sort of like this guy would ask us, “How does it make you feel?” Blah, blah, blah. “What other things feel like-- what other things make you feel that way?” Blah, blah, blah. And people were just yelling random words. And it was kind of like in two phases. One was yell random words and then the other one was kind of like, order this list of random words on the whiteboard. And then, you know, we took this list of words, we sent it to the lawyers and said please go to top to bottom, the first one that passes your test we will go with that whatever it is. And it was lucky number four.

Hsu: Lucky number four Java.

Gosling: Yeah.

Hsu: The coffee one.

Gosling: Yeah. Well, number three which was my favorite was Lyric and it was so hideously taken. And number one, the one that most of the people in the project liked was Silk. I personally found that kind of creepy because it just so reminded me of spiders. Even though people who liked it said, “Oh, it’s the Web." It's like yeah, but it’s spiders. “No, it's the Web.” No it’s spiders. But we were not a very big group and I'm into democracy. So then I forget what number two was but number four was Java and we went with that.

Hsu: I want to go back to talking about the browser strategy. So Java applets would run as plug-ins in the browser?

Gosling: Yeah.

Hsu: So at the time that was the story was that the user would need to download an applet to run on the browser.

Gosling: Well, the applet would be automatically downloaded for them and then run inside a secure environment.

Hsu: So why didn't Java applets succeed in the long term?

Gosling: I think the biggest problem was how incredibly difficult it was to integrate stuff into Web browsers particularly in the nineties. The set of bugs that we had with integrating into Internet Explorer, this is Internet Explorer on Windows 95 and trying to make anything that was reliable and secure was really difficult. And trying to-- people would find security holes which were essentially always bugs in the integration; sometimes they were sort of leveraged attack bugs. So there was a whole class of these where if you had managed to corrupt a DNS server in a certain way then the Java VM-- so early on the Java VM was very trusting of the domain name servers. But there were a lot of things where if somebody had broken into a domain name server, corrupted the domain name server, then you could write an applet that would exploit that corruption. And so it's like stuff that was outside of us but-- and beyond our control but we could make it a lot better by changing the way that we trusted the domain name server. But there were all kinds of attacks based on bugs in the Windows integration. The direct integration into Windows Explorer-- or Internet Explorer was just a nightmare.

Hsu: Was it, also, the case for Java on the desktop? What were the difficulties of that?

Gosling: That was quite a different story because Java on the desktop actually worked really well. The criticisms that we got most often were somewhat philosophical. So the sort of standard thing that caused difficulties was we worked really hard to make it so that if you had a desktop app it could run on the Mac. It could run on a UNIX box. Or it could run on Windows. But all of those boxes had their own very specialized weird services. And it took quite a long time to get things like cut-and-paste to work so that we would have a consistent API across all the platforms. Some of the ones like "how do you talk to a USB device", because the USB device support on all those platforms was so radically different. The USB interoperability never really happened in the core platform. There evolved a number of libraries that did it. But in some sense it was our sort of religious sticking to the "write once, run anywhere" thing that for enterprise applications, because all these platforms really did look the same. It worked really, really well. But on the desktops where there was a lot of peculiarities from one desktop to another, things got really difficult. So with-- when Apple came out with OS X they did everything using anti-aliasing. And Microsoft did everything with <inaudible 1:16:20> square pixels. And people would run-- write software that would just sort of have a deep knowledge in the way that square pixels worked on Windows and then yell at us when they didn't work with anti-aliased pixels on the Mac. And, you know, if you actually read the Java spec and you used the various primitives like drawing a line correctly then it worked just fine on all these platforms. But people would often exploit sort of weird knowledge that the properties of the square pixels leaked through in things like the way that coordinates got rounded and things. So some of those things just got difficult. And on any one platform life was pretty easy. And people wrote lots of desktop apps. But, in general, desktop apps have kind of disappeared from all the platforms. It's really kind of crazy the extent to which Web apps really took over the universe. And Java became *the* platform for driving Web apps. And I think that-- I find it particularly weird these days that it's considered religiously distasteful to write a desktop app that runs on a laptop, say; and yet, it's the height of religious excellence to write a cellphone app that winds itself into the properties of the cellphone. And if you actually tried to write cellphone apps that are entirely done in the browser they fail miserably. It's just a horrible universe if you try to do that. And so people have this strange dichotomy about desktop apps versus mobile apps. And,

you know, this whole religion that desktop apps must suck, it just disturbs me. I see people write developer tools as Web apps and I haven't seen one of them that didn't suck. I mean at best they look like vi in the eighties. And I'm a big believer in fancy IDEs like NetBeans or Eclipse or IntelliJ. But the cult of vi I find bizarre.

Hsu: You mentioned how successful Java has been on Web servers. Talk a little bit more about why it's been so successful on the server-side and also in the enterprise.

Gosling: It's been a whole lot of different reasons. Things like it abstracts away a lot of the peculiarities of the platforms. It gets rid of a lot of the mysteries of failures. So in C there is a broad category of bugs called memory corruption bugs where the problem will occur but effects from that will be manifested far later. And Java makes large swaths of memory corruption bugs impossible. Or at least they get detected closer to the point where the problem really occurred. And things like array subscript checking, the model for arithmetic, they all make a lot of things more reliable, more easily. Then there are things like multithreading. C for the longest time had no way to do multithreading. It still only kind of has multithreading. If you try to write a multithreaded app in C, you're in for a world of hurt. And when you've got any of these large server class machines they've got a lot of CPUs, they've got a lot of memory. And if you can't run multiple threads, you are going to die or, at least, you are not going to get out of your machine what you want to get. Similarly, these machines all have a lot of memory. And Java, with its garbage collector-- and it sort of abstracts away the notion of what a pointer is so that running on a machine with a couple of gigabytes of RAM versus a couple hundred gigabytes of RAM, the Java programmer doesn't know, right? Whereas in the C world the transition from 32-bit addresses to 64-bit addresses is majorly traumatic. And, you know, a lot of apps in the C world are still struggling with that. Whereas with Java apps, if you run them on a 32-bit machine, fine. If you want to run that on a 64-bit machine, the porting effort is zero. So, it just runs. Doing storage management in C, you know with small scale apps that don't run for very long, you can commit all kinds of sins in storage management and you're fine. Once you're building apps that need to run for hours and days and weeks and months getting storage management really solid is crucial and the garbage collectors are absolutely essential to making that sort of large-scale app really run.

Hsu: And I also want to talk about sort of the other side of things. So, Java's also been very successful in small devices, embedded devices, Internet of Things. In a way, this has sort of gone back to the original sort of purpose that Java was designed for.

Gosling: Yeah, so, there are lots of embedded devices out there that use Java on them, particularly ones that get more complicated. You know, when you're doing small, simple things, all the facilities in the JVM are probably not all that compelling, but once you're doing a complex device, you know, particularly if you have to do any kind of, like, planning processes and things-- you know, if you have to do modeling of the situation around the robot and you have to do collision avoidance and that kind of thing, you need very complex data structures and they need to evolve in fairly chaotic ways. And a system like Java is fabulous for that.

Hsu: Could you talk a bit about Java's success on mobile phones?

Gosling: So, the mobile phone success, it really started with NTT DoCoMo. And they had exactly the problem-- well, kind of a flavor of the problem that we had heard of in our early feedback in the Green Project where there were-- you know, people wanted to be able to use different processors. And, at the time, DoCoMo was kind of a government monopoly. They've kind of separated out and they were-- they had various mandates and, so, they couldn't-- so, they had to work with a lot of different handset manufacturers. And they made most of their money off of network bandwidth and one of their goals was to do things that would drive bandwidth usage up. And, so, one of the categories of things that they really got excited about was to be able to play games and things like multi-player games or just simple board games like backgammon and checkers and chess, but not necessarily playing the game with the phone, but playing a game with a friend of yours. And if you have to do, like, a whole ecosystem of different handsets, then if you've got a software developer who's building something like a backgammon game to do it in an ecosystem of handsets, you either tell every software developer to build a dozen versions of their backgammon game or you get the handset manufacturers to all agree on a common software platform. And the DoCoMo folks decided that it was easier to mandate a software platform than it was to get all of these software companies to build a dozen copies of everything. So, they did that and it was-- they did the very first app store and because their goal was to increase bandwidth usage, they did a revenue split that was 70-30; namely, the software developer got 70 percent and they got 30 percent. And the 30 percent, it covered the running of the app store and the security checks and that sort of stuff, but it wasn't viewed by them as a really significant revenue source. Their revenue source was bandwidth and this was dramatically successful for them. And when talking to people in phone companies and the rest of the world, and particularly in North America, you know, one kind of conversation that I had a bunch of times is ones where I'd say to some phone company executive, go "What do you think of these guys at DoCoMo? They're making a lot of money by doing-- by having this app ecosystem." And after DoCoMo did it, a lot of other countries did, but the U.S. was very resistant to that. And when I would ask this question to North American phone company executives the average response was, "Those guys at DoCoMo, they were such idiots. They left so much money on the table. We've had our crack team of phone company analysts figure out what applications people need. There's like a dozen of them. We'll just have some of our phone company engineers build those apps and we make all the money. And it's like, "Good for you." And then, you know, when Apple did the iPhone, you know, their app store, their revenue slice [ph?], everything pretty much mirrored what DoCoMo did.

Weber: Are you talking about i-mode or separate from i-mode?

Gosling: Well, this was starting with the i-mode. Yeah, they evolved quite a bit, but it really started with the i-mode stuff and they actually sold i-mode outside of Japan in a few places.

Weber: So, i-mode used Java.

Gosling: Yeah.

Weber: It was compact HTML, but it was Java. Okay. Java for the applications.

Gosling: Yeah. Well, the pure sort of compact HTML stuff, the ones that had the-- those ones were pretty miserable and ugly when they-- they did kind of next version up where they did-- one of the things that was a problem with DoCoMo was that there had been this international group of engineers from phone companies outside North America who sort of came together to come up with this Java ME spec. And DoCoMo was so rabidly eager to get going that they launched before the spec was finished. And, so, the thing that they launched with was, like, an early snapshot of the Java ME spec. So, once they sort of up-leveled from the original, just like, pure simple HTML which was-- well, it wasn't really even HTML. It was really primitive. That was when things really took off for them. Because as soon as you can do-- I mean, fundamentally, once you could do games, then the flood gates opened. You know, and you'd ride the subway in Tokyo and people would be there playing backgammon with their friend who's on another train going someplace else.

Hsu: Yeah. Could you talk a little bit about Android?

Gosling: Could I talk about Android? <sighs>

<laughter>

Gosling: There have been so many lawsuits over that and so much misbehavior over that-- and I have really conflicted feelings about Android. You know, on the one hand, the stuff they did was fundamentally lovely. How they, and in particular the guy who led the effort, went about it was absolutely despicable. And, yeah, decades of litigation and billions of dollars in lawyers later-- I think I should just shut the fuck up.

<laughter>

Hsu: Okay.

<break in recording>

Hsu: Okay, what are your thoughts about the Oracle acquisition and the Google-Oracle lawsuit?

Gosling: Yeah, I don't know how to talk about Oracle without starting to scream and yell and froth at the mouth. You know, they have a well-earned reputation as a company that carries knives and swords at all times. And, at the same time, the way that the Android project got started and some of the games that the leadership of that project engaged in were pretty despicable. There's no good side in that particular fight. I think some of the stuff that Oracle is pushing for in terms of legal precedent, if that really happened, then it could be a huge problem. But, at the same time, you know, Google's behavior was not great either. So, not a fan of either side.

Hsu: How open was Java in the beginning? It wasn't made fully open source until, like, 2005-ish?

Gosling: Yeah, so, all the source code was available from Day Zero. We went through a variety of licenses for that source that-- the issue for us was that there were a lot of people in the community who were just wonderful, right? And with whom-- for whom an open source license would be just fine. There were a small number of large, powerful actors who are on the-- you know, give them an inch, they take a mile. And people were-- you know, a bunch of these large actors were attempting to play all kinds of nasty games. I mean, I was never in the middle of the nasty games, but I was kind of on the edge of it and it was pretty freaking appalling. And the license terms kept evolving around what we figured it was sort of safe for us to go to, because a bunch of these large bad actors were-- their whole goal in life was to crush us flat. And we didn't want to be dead. And we were sort of caught in this really difficult position between wanting to work with the community and wanting to not be dead. And it wasn't like it was-- you know, could we outcompete people? It was just literally large actors playing ugly games and it's never a good place to be.

Hsu: So, how happy have you been with Java's success and legacy?

Gosling: I've been ecstatic. You know, we're now nearly 30 years since the first Java program ran. And it's really hard to do anything in modern life without, somewhere in that chain, you're interacting with a Java program. You know, a lot of large websites out there are big bags of Java code. You know, certainly, if you use-- the list is gigantic, but it's very, very commonly used. And, yeah, how could I not be thrilled?

Hsu: What do you hope and/or fear the net will look like 20 years from now and Java's role in it?

Gosling: So, you know, the way I've always mentally phrased it to myself is "What future do I want to live in: 'Star Trek' or 'Blade Runner'?"

<laughter>

Gosling: And in the network that I would love to be in 20 or 30 years from now is one where security and privacy and individual rights are respected. One of the big pushes in the design of Java was to make sure that it was the sort of thing where you could do a much better job of building a set of secure systems. And I think that's very much been true. I mean, it's-- you know, if you have to build secure systems, Java is pretty much your best bet right now. And yet there's a lot of stuff that's very much tilting the universe in the "Blade Runner" kind of direction. I mean, you look at the way that people abuse platforms like Facebook and Twitter and it gets pretty ugly. And I would like that to not get worse. And I would love it to get a lot better.

Hsu: How do you think technology will impact the lives of your children and grandchildren?

Gosling: Well, you know, the lives of my children have been completely different than mine. I mean, I grew up without a cellphone and with-- you know, so, it's this double-edged sword, right? On the one hand, they have access to an inexhaustible selection of knowledge. I mean, I thought it was really great when my daughters-- each of them went through phases. I mean, there's kind of like a grade in school where all of a sudden Wikipedia becomes a game. And kids learn—it's like, okay, you pick two random

concepts and you go to the Wikipedia page for the first one. Find a path through Wikipedia links that gets you to the second one. You know, so, how can you get from global thermonuclear war to platypus? And that one thrilled my kids for the longest time. And, on that, that kind of thing is just fabulous. On the other hand, the sort of shaming that happens in Facebook is like a-- it's an epidemic. The way that people-- you know, it doesn't even have to be evil. It's just like on Facebook kids post the good things in their lives, but never the bad things. So, when kids look at their friends' lives on Facebook, they don't see any problems. So, then they have their own problems and they go, "I'm all alone. I'm the only one in the universe who's fucked up." Whereas if you have face-to-face relationships with friends, you know, the good and the bad comes out and you get a more balanced view. And Facebooks kind of selects for bias. And that gets really, really difficult.

Hsu: What advice would you have for young people starting out in your field?

Gosling: In my field? Read a lot, build stuff, have fun. I mean, that was kind of my formula. And I think it's still a pretty good one.

Hsu: How do you think cultural institutions like the Computer History Museum can foster positive social change?

Gosling: You know, an institution like the Computer History Museum, the thing that they can uniquely do better than anybody else is provide perspective. You know, you just asked me about the contrast between my childhood and my children's childhood to sort of highlight that perspective and that change can be a really important point for a lot of institutions. And even something that's not the Computer History Museum, like, an art museum-- and San Francisco has a lot of excellent art museums-- there's a dramatic difference between seeing one of Monet's water lily paintings on a screen and seeing it in real life. Or the difference between seeing a picture of a tree and a real tree. And if we can get people to be less sucked into the screen-- I mean, the screen is a wonderful thing. You can do great stuff with it, but it's become really, really out of balance. And my youngest daughter right now is on an extended sort of adventure trekking trip and one of the big features is that she's spending three months almost entirely out of reach of any kind of cellphone signal or even power. And I'm really curious what she's going to be like when she gets back.

Hsu: So, James, tell us when and where you were born.

Gosling: I was born May 19th, 1955, at the Grace Salvation Army Hospital in Calgary, Alberta.

Hsu: And tell us about your family.

Gosling: Let's see. My mother's name is Joyce Morri-- [ph?] Joyce Morrison [ph?]. My dad was Dave Gosling. My mom grew up in Banff. My dad grew up just west of Calgary on a farm near Dalemead, Alberta. We lived in Calgary-- you know, my father's side of the family-- in the early years, he was the only member of the family that moved to a city. And I had one sister, Barbara, and one brother Jeff. And my

brother's an industrial designer. My sister and her husband, they run a consulting company that does computer software.

Hsu: What's your original ethnic background?

Gosling: Well, so, my four grandparents are English, Welsh, Scots, and Icelandic. And if you-- like, if you called my Scots grandfather English, he would kill you.

<laughter>

Gosling: And my-- yeah.

Hsu: What were your parents' occupations?

Gosling: My mom was a school teacher. My father did an assortment of things. He could never quite figure out what he wanted to do when he grew up. Mostly what he did was he was kind of a tour guide for geologists. So, if you were an oil company and you wanted to do exploration, expedition, up on Baffin Island or in the Yukon, or something like that, and you wanted a bunch of geologists to do their thing, my dad would put together the expedition. He would hire the pilots, rent the helicopters and the planes. He would hire the laborers and the cartographers and the cooks and he would arrange all that stuff and then he would run it in the field. He did that more than anything else, but he was kind of all over the map.

Hsu: What sorts of values were you raised with?

Gosling: Hard work. Nobody was particularly wealthy anywhere. Nobody was particularly poor. One of the things that I later realized was peculiar was in my extended family, there was essentially nobody who had a career. I never had this view that I need to get the next promotion and getting promoted was never a thing for me, because in my extended family it was kind of like, well, you had a good year, you had a bad year? You know, depends on what the rains and the snows bring, you know, what the markets are like-- you know, did you plant the right crop? Did you breed the right cattle? Did you make-- do the right thing in whatever? And, so, I've always just valued the work, never the career trajectory.

Hsu: What sorts of-- were there any political or religious leanings?

Gosling: Well, sort of on the religious side of things, my parents were both sort of technically protestants of some flavor. I guess you would call my dad Anglican, kind of. We weren't practicing anything really. One of the peculiarities of Alberta was that it had a lot of religious groups of various stripes. You know, Mormons and Hutterites and Mennonites and Old Believers and all kinds of strange people. Doukhobors. And, you know, so, from a religious point of view, I never had this view that religion was a homogenous, unified thing. It was always this chaotic mixture of different people with different belief systems. And they were all fine. I mean, some of them were a little weird. I mean, the fact that if the Doukhobors got pissed off at something, they would strip naked and walk down Main Street. That was kind of their form of protest. And that was always a little weird. But, otherwise, it's like-- and, also, Southern Alberta, like most

of Canada is just a patchwork of Indian reservations. There were a lot of tribes around us and got to know a number of them reasonably well. My dad hired Indians all the time and a lot of my relatives, you know, for farming jobs. I remember one of my dad's classic lines was, when I asked him why he hired so many Natives for his jobs in the Arctic, his sort of stock answer was roughly, "Any round-eye that wants to work in the Arctic is either drunk or deranged." You know, so, my dad wasn't a drunk, but-- so, he defined himself as deranged. But--

<laughter>

Gosling: Yeah, so, you know, it was a very mixed world. You had to get along with everybody, because-- and to understand what was interesting about everybody and what were the hot buttons to stay away from. And, as far as politics go, mostly it was pretty benign, except that there was a lot of antagonism towards Eastern Canada. There's a sort of long history of Eastern Canada rigging various regulations. And especially if you were a farmer, there were these things called the crow rates where the politicians on the East Coast had kind of rigged all the freight rates to make it so that it was essentially impossible to do any manufacturing in Western Canada. You know, it was cheaper to ship a live cow, for example, than it was to ship a slaughtered cow, even though the actual cost of shipping a live cow was much higher than a slaughtered cow. So, you couldn't-- it was very hard to build up much industry, because the rates really favored sort of the primary products rather than anything you would manufacture out of them. And people, like my dad, were very cranky about that. And that was the only thing that was, like, a big theme that ran for years. But, mostly, you know, this is Canada, right? People view government as more of a comedy than something to think of. I mean Canada got started where the very first prime minister was an absolutely unrepentant alcoholic, only accidentally seen sober. You know, that seemed to breed an attitude of sort of humorous skepticism.

Weber: But the outdoors was a big part of your father's life and I think you said your mother had been raised in Banff and involved with the outdoor industry. Were the outdoors important to you?

Gosling: The outdoors were pretty important to me. I did a lot of hiking and skiing. I mean I couldn't avoid skiing, because my family-- my one uncle was on the Canadian Olympic Ski Team. My grandparents had for many years managed a small ski resort. They had-- they owned a small collection of cabins in Banff. And I remember going to my grandparents' 50th wedding anniversary party and part of that was people standing up and telling stories about my grandparents. And it was clear that my grandmother, who was Icelandic, was pretty epic. You know, when I was a teenager I would go skiing with my grandmother and it was very clear that there was zero chance that I was ever going to be anywhere near as good as my grandmother. You know, I'd get down to the bottom of a hill, I would have worked my ass off negotiating this field of moguls and I'd just-- <mimics panting>-- and my grandmother would be standing at the bottom and she'd look at me and she had a series of stock lines and the one that she used the most often that annoyed me the most was, "You're breathing hard. You're doing something wrong."

<laughter>

Gosling: Because for her, skiing was all about balance. And she-- it wasn't about muscular exertion. It was about just guiding your skis by shifting your weight. And-- yeah.

Hsu: What was your neighborhood in Calgary like?

Gosling: It was pretty middle class, bungalows-- really homogenous. There was a slightly well-to-do neighborhood, kind of uphill from us, but compared to, like, the-- you know, when you go around the San Francisco Bay area the sort of financial skew between families is really dramatic and it was nothing like that. I mean, we had a three-bedroom house and the well-off people had, like, maybe four-five bedroom houses, but lot sizes were about the same. They were all pretty similar. It was kind of hard to tell. There were no private schools. When I moved to the U.S. when I was 22, the whole private school thing seemed kind of weird to me. And the whole, like, private universities versus public universities also seemed really weird to me.

Hsu: Describe your childhood.

Gosling: I was a pretty geeky kid. Actually, I was a ridiculously geeky kid. We never had money for me to do the usual sort of geeky things. I could never afford to buy connectors or wires or parts or anything like that. So, all of my geek supplies were obtained by digging things out of other people's trash. It's amazing what you can do with the guts of an old television set or if you go in the back behind the phone company's switching center, you find all kinds of useful stuff just in the trash. Did a lot of that. I did not get involved in sports. My eyes were really bad. So, growing up-- well, and I'm also-- I have strabismus. My eyes don't align right. So, if I don't have glasses I see two of everything. So, when I was a kid, you know, if somebody threw a ball at me to get me to catch it, it was a random toss-up as to whether I would reach for the right ball. You know, there would be the real ball-- and actually often what would happen is that the real ball would be here, but I'd see two images there and there. And, so, you know, I was always viewed as a complete clown. And it wasn't until I was seven, maybe eight, that somebody thought that maybe they should check my eyes. And I remember getting my first pair of glasses and walking out of the doctor's office and I could see the-- you know, I had seen, like, telephone poles all over the place, but I could never see the wires. I had never known that these big wooden sticks that stuck out of the ground actually had wires in between them. And I remember walking out of the optometrist's office and all I could see was this maze of wires all over the sky! And that was really fascinating. And the optometrist's office was about a mile and a half from my parent's place and I had to walk home and I just remember that walk home-- just, like, looking at all these wires and following the wires from here to there and it was just the coolest thing ever. But one of the things that came from that was from going through kindergarten, first grade, and most of second grade with being damn-near blind was learning to read rather strangely. Right? So, every word-- I didn't really get the concept of letters, because if you have a word, the letters all kind of like smoosh together. And, so, I could kind of learn the-- I kind of learned that the words were blobs kind of like this. Yeah, one of the consequences was atrocious spelling. You know, the-- if you blur them out, an "O" and an "E" and an "A" all look kind of the same. "D"s and "B"s look the same. You know, if it wasn't for little memory tricks, like the word "bed", I'd still have a hard time with "B"s and "D"s. But that also was also somewhat isolating, right. And I could never-- I never figured out the attraction of team sports. I was just-- thought that was just kind of goofy.

Hsu: What sorts of hobbies did you have?

Gosling: Well, the primary hobby was building stuff. My dad had a little workshop in our house. And then back at my Uncle Melford's place, which had been my grandfather's place, he had a big workshop, but it was the typical kind of farmer's workshop. And because it was my-- you know, it started out as my grandfather's workshop. It had all kinds of weird things, like a full blacksmith's kit. You know, a blacksmith's forge and anvils and big hammers; and if you want to make chain, you can make chain. And while my Uncle Melford never used any of that, I remember my dad showing me how to fire up the forge and make chain and do simple stuff there. But, you know, I was just always building stuff, whether it was little model airplanes or, you know, once I got onto building sort of electronic stuff-- yeah, I was always just making stuff.

Hsu: What sorts of literature and media did you consume?

Gosling: Well, we did have a TV. I watched a moderate amount of it. Not huge quantities of it. I read a lot. We were about a mile away from a public library. So, I went there and discovered science fiction and would just-- you know, by the time I was in fifth and sixth grade and I had finally learned to read properly, I would just read gobs and gobs of science fiction. And I'd try to, you know, learn was there much reality behind it and that turned into me reading an awful lot of non-fiction. At some point, my parents bought a copy of the World Book Encyclopedia. And, for reasons that completely escape me now, I read the entire encyclopedia from beginning to end a couple of times. And my dad had this old, old copy of the Encyclopedia Britannica and I actually read that from cover to cover at one point even though it was really out-of-date. And I have no idea why I thought that was a cool thing to do. But, you know, I sort of did that.

Hsu: Describe your school experiences.

Gosling: You know, school experiences-- I mean, the schools that I went to were-- you know, culture-wise the top of the heap was always the jocks. And I was probably the least jock-like person you could imagine. And a lot of the time I was really like the only nerd. You know, so, I was certainly at the bottom of the social structure. But, you know, mostly there wasn't much that was terribly dramatic. Mostly just going to classes. At one point, when I was in junior high school, which in America is middle school, I-- well, my dad took me over to the University of Calgary, which was, like, two miles away, because a friend of his worked there, and took me on a little tour. And I kind of fell in love with the place. And, so, starting at the-- like, for the last year of junior high school and all of high school I pretty much lived at the university. And, so, the university in a weird way kind of became my family really early on.

Hsu: So, at 15 you were working with the physics department.

Gosling: Yeah, yeah. So, I had started-- you know, when this friend of my dad's took me on this sort of like tour, one of the places we went through was the data center. And, you know, spinning tapes and flashing lights and stuff that just dazzled me. It was just like the coolest stuff ever. And, you know, it was only a couple of miles away. I could get over there pretty easily. You know, I could get over there on my bicycle or I could walk in the middle of winter. And I was a big kid, so I kind of looked like a university

student. And they had a special library for the computation center that had all the manuals. And I just read stuff. And I didn't exactly have a whole lot of security on anything. You know, people would have, you know, their card decks with their school assignments and one of the first cards in the deck would have your user ID and password on it in clear text, which is these days just insanely goofy. But all it meant was that you just had to pick somebody's assignment deck out of the garbage and you had their credentials. So, I taught myself how to program in Fortran and PL/I. And they had some small little mini-computers. They had some PDP-8s. I taught myself how to write programs on the PDP-8. And one of the folks that I got to know who was a student, he figured out pretty quick that I was not a student and that I was doing pretty good writing software for this PDP-8. And since he worked for the physics department and they needed some people who could write software, he said, "Do you want to try writing some software for the physics department?" And I said-- well, he actually took me to one of their labs and I was just like, "Yeah!" And, so, I started working with this one group and I pretty much spent all of high school spending every hour I could writing software for the ISIS-II project in the physics department.

Hsu: That as a satellite?

Gosling: Yeah, that was the-- right, the ISIS-II satellite was the one that I was helping write analysis software for. And that group carries on. They still do a-- they still have a very strong upper atmosphere physics program. And there was a few people whose background was writing software, but mostly it was a bunch of physicists. And, so, I was Code Monkey for a bunch of scientists. And I really liked that and I really got to enjoy the kind of "Code Monkey for Scientists" thing. And got to know a bunch of the physicists and just had a lot of fun.

Hsu: So, it was natural for you to continue on to attend the University of Calgary when you graduated high school.

Gosling: Yeah, and I started doing other jobs with other groups and I had a lot of part-time jobs working for various projects around the university and then when I started-- when I became a student, I just-- the computer science courses were-- you know, it was like, "Don't bother showing up to the courses, just do the exams and do the assignments." You know, the assignments would take no time at all and then I could spend all the normal homework time just working, because working was a lot of fun.

Hsu: You mentioned that you hadn't actually taken the courses required by the university to graduate, but you somehow got out of that.

Gosling: Yeah. That was-- so, that was sort of a funny incident, because I get to the end of my undergraduate. I had applied to a bunch of grad schools. Carnegie Mellon had accepted me and I go to Carnegie Mellon. And then several years later somebody told me this story, which I then went, like, "Really?" And it was corroborated to be true was that I hadn't taken nearly enough of the sort of elective courses, things like anthropology and English and those things. I had taken way too many computer science courses. And, so, technically, I was not allowed to graduate. And a letter to that effect was sent to the head of the department, who at the time was a guy named Anton Colijn, the crazy Dutchman who always dressed very properly. And he apparently just exploded and made quite a scene at the registrar's

office and got me graduated. And nobody told me until a fair number of years later. It was like, “Wow. Good on you, Anton.”

Weber: Do you want to tell maybe, just to give a flavor of the life there, maybe one of the pranks that you were involved with, with some of your professors, I believe. Just choose one.

Gosling: Yeah, so, the-- for reasons that I don’t really know, there was quite a culture of pranking and particularly amongst the professors. And sometimes it was a little hard to tell who was the ringleader of certain pranks, because people would be getting roped in. But, like, Mike Williams, who was involved in the Computer History Museum early on, he had a rather extensive collection of old computers, things with drums and vacuum tubes and all of that. And a lot of that was stored in the basement of the Math Sciences Building where the data center was. And there was-- that basement had been kitted out for classes and things and then used for the data center. So, it didn’t have nearly as many people in it as it normally would. And, so, it had way more bathrooms than it needed and, in particular, there was a rather large women’s bathroom, kind of like in a back corner, that was never used. I mean, it was-- and, so, Mike stored a lot of his stuff in this really large women’s bathroom. And, at one point, we did the obvious office-and-bathroom-stall swap between one of Mike’s not exactly enemies, but frenemies, a guy named Norm Barnecut [ph?]-- moved Norm Barnecut’s office into the bathroom stalls and moved some bathroom stalls into Norm’s office and re-arranged some of the signage. On the prank scale, that was a relatively minor one, but that one was a lot of fun.

Hsu: So, you went on to graduate school at Carnegie Mellon in 1977?

Gosling: Yeah.

Hsu: And it says that you actually got into Carnegie Mellon as one of the sort of special students at the discretion of the department head?

Gosling: Yeah. I mean, when I got there I sort of figured out that something was a little odd, because I was the only student in my class who was paying tuition. Everybody else was there on a RA. And after a couple of semesters, I got on an RA. But when I did my thesis defense, you know, one of the traditions for a thesis defense is that you always buy a keg of beer and it goes into the department lounge. And no matter how your thesis defense goes, beer is appropriate afterwards. So, after my thesis defense, you know, we’re sort of in the lounge and I had just graduated and one of the faculty members came up to me. And he was one of the guys who was often on the admissions committee. And he told me this odd story, which was that the grad school at Carnegie Mellon had always received just dramatic numbers of applications. You know, several thousand applications at the time for 15 positions.

Hsu: This is in computer science.

Gosling: Yeah. And the computer science department at the time only had one degree program. They only did a PhD. They didn’t have an undergraduate program. They didn’t have a master’s program. They just had a PhD program. And, so, the admissions committee had this difficult job of going through

thousands of applications and trying to figure out who to admit. And they would do some cursory passes that-- there's-- some of them are just obvious from things like GPAs and backgrounds and that kind of stuff. And then they would sort of do multiple passes over them and the passes would get sort of more and more subjective. And as it got down to picking the final 15, you know, it's like reading tea leaves. And it was a pretty uncomfortable process. And they had this-- there was this sort of theory that the hard part at the end of the selection process was really kind of pointless, that if you had taken the top 100 and picked at random, anybody that you picked would be fine. And, so, the department apparently had this tradition for some years that they would go through this process and then at the end they would randomly pick one person. And at my graduation beer bash, I got told, "Congratulations, you're the first randomly selected student to ever graduate."

Hsu: Wow.

Gosling: And I've told that story a few times and then several years later, the guy who was the head of the department, he visited me and we had this conversation, he said, "Well, it wasn't quite random." And then he describes this process that reminded me more of a Ouija board of him having these stacks of things on his desk and he'd just have to randomly pick something that was outside the usual. And it's like, okay, dice or a Ouija board. I mean, it's whatever.

Weber: But randomly picked from how big a pool? From the 100?

Gosling: From the 100.

Weber: Yeah.

Gosling: Yeah. And when the first guy sort of explained this process to me, one of the things that he said was that one of the challenges that they have in these selection processes when you get to the end, it ends up being about letters of recommendation and that kind of stuff. And if you're a kid from some nowhere university, which the University of Calgary certainly was at the time, nobody in the admissions committee would know any of the people writing these letters and they really wouldn't know any of their backgrounds. So, they wouldn't have any calibration on-- you've got this letter of recommendation [that] says glowy, rosy things, because that's what letters of recommendation always do. But it's like, so, who wrote this letter and how credible are they? And when you're a kid from nowhere, they have no way to know. Right? They don't know anything about, like, the quality of the program that you went through. I mean, sure you get this GPA, but a GPA from here is different from a GPA from there. And when they don't know anything about where you came from, there's-- it's really hard to judge. And, so, yeah, they had been running this experiment and I don't know if anybody since then has succeeded, but it was decidedly odd. I sort of feel like there are a bunch of these places where I've kind of won the lottery.

Hsu: So, you were the only one out of that or out of your cohort to have been selected through that process or was--

Gosling: Apparently, I was the only one.

Hsu: Okay.

Gosling: Apparently, they did one a year.

Weber: But of those, each year you were the first to graduate.

Gosling: Yeah. In whatever number of years they had been doing this one-a-year thing, I was the only one who graduated.

Hsu: And, so, how did you get involved in the school's migration effort to Unix?

Gosling: Well, so, one of the-- so, when I was at the physics department, the guy who originally met me and hired me in there, he was a guy named Bob Sidebotham. And Bob kind of went on and one of the next jobs he did was working for the faculty of environmental design running one of their-- they had a few little computers. And he-- you know, there was this cool thing that came by that was Unix. So, he installed Unix on this PDP-11/40 and, so, I started playing around with this PDP-11/40. And that was like the first computer that showed up on campus that was-- and this was '76-ish, '75. So, I got to play around with it and get to know it somewhat and then I got to Carnegie Mellon and at the time most of the machines in the CS department were PDP-10s. There were some PDP-11s, but they were mostly running homegrown operating systems, which made them kind of hard to use, because they were mostly just-- you know, by "homegrown" I mean, some PhD thesis student's project that worked well enough for them to graduate, which is never like well enough to actually do anything. And, so, there were a couple of us that got involved in running more useful things on these PDP-11s. And because I had been involved a little bit in Unix before, I then got involved in running Unix on these machines. And then eventually the department started installing VAXs and there was a large lobbying effort from the folks at Digital Equipment Corporation to get us to run VMS. And most of us, who-- particularly me. I mean, I was kind of like a leader of this and I'm reading the VMS manual and going, "You're kidding, right?" whereas Unix seemed to be way more sensible. And then the Berkeley folks started doing all of their work. And, so, it was like, "Yeah, we've got to do that," and there was this lobbying between the sort of VMS crowd, which is mostly driven by folks at DEC, and then the whole sort of Internet community. You know, I was definitely writing a lot of impassioned emails, you know, some very nerdy emails, some-- they got-- they were kind of all over the emotional spectrum, but, because, fundamentally Unix did the job and Unix had all the flexibility that we needed. And one of the early projects that I did was porting one of the early versions of BSD Unix to a multiprocessor made out of PDP-11/40s. This was a machine that had been built at CMU and so I ported BSD Unix to this multi-processor in, like, '78, maybe '79. I forget exactly. And it was great. I mean, but then, of course, it's within weeks of getting this port stable enough that I could do my day-to-- daily work on it. The project who had built this machine decided that, yeah, they were done. They decommissioned it and ripped it apart.

Weber: But you mention email and Internet. I mean, back in Calgary, when did you-- had you been interested in networking per se earlier on?

Gosling: So, networking at the University of Calgary was just about non-existent. There were a couple of people in the data center who had some contracts to do some of the early, you know, X-dot-whatever protocols that were always, you know, strange serial line stuff that was mostly not terribly well thought out, because-- so, they were sort of hired to implement some of these standards. None of it was ever useful at all. Networking was always sneaker-net. It was pretty much always big mag tapes. It wasn't until I got to Carnegie Mellon that I encountered real networking and at the very beginning it was all ARPANET. And, eventually, we got some of the early Ethernet-- you know, the big fat coax cables that you sort of tap into. And I remember somebody from Xerox giving a talk where they said, you know, "Ethernet's really, really fast. It's like three megabits." And then he sort of paused for dramatic effect and said, "Well, it's really not three megabits. It's 2.95 megabits." And he said, "So, nobody really worries about me having rounded 2.95 to three, but that rounding error is exactly the same as the speed of the fastest link on the ARPANET," you know, because it was like 50 kilobits was the fastest link on the ARPANET. And I was like, "Okay." And I got involved in doing a bunch of protocol implementations for Unix and the whole early time when people were debating what to do to go on from the ARPANET protocols and the debate that ended up with TCP. I was never in those debates, but one of the people who was involved had sort of bamboozled me into implementing one of the sort of technology candidates. So, in Xerox there was this pair of protocols called PUP, the PARC Universal Protocol, which is kind of like IP, in fact, almost exactly IP; and BSP, their Byte Stream Protocol. And so I did an implementation of that protocol pair for Unix and did the sort of protocols above that to get to talk to, like, Xerox printers and Xerox filers and that kind of thing. And one of the reasons that I had gotten convinced to do this was to validate the spec document and it mostly worked. And, by mostly, I mean, it was like one of the weirdest bugs of my life that it would work for months and then once every couple of months a link would hang up and I never found the bug. And then I-- after I graduated, I go to Carnegie Mellon and I'm at Carnegie Mellon for a couple of years. And I get a message from one of the folks from back at CMU that was like, "I found the bug," because I guess they had just gone completely nuts trying to find out what this bug was. And it turned out that there's this matter of semantics about what is a bug. And, so, I had actually implemented the spec correctly. And there are parts of that spec that are about how you do error recovery. And it turns out that the original Xerox implementation that ran on things like the Dover printer and the filer, there was like one little nit that they did differently than what the spec said. And it was a really minor nit that only bit you if two packets in immediate succession both failed. If you just lost one, it was fine. If you lost two, but there was a good packet in between them, it was fine. If you lost two packets that were immediately adjacent, it would lock up. And that was just because I implemented the spec. The other code was written before the spec. There was this fine nuance that people hadn't really recognized. And, so, the spec got backed out and my code got jiggered to match the Xerox code. And that was, like, quite annoying.

Hsu: And you co-wrote the bundle program on Unix? I have that somewhere?

Gosling: Bundle?

Hsu: Never mind, we can move on.

Gosling: You mean there was a-- Bundle? I don't-- I never wrote a program called Bundle. I wrote it, I mean, one of the little things that I wrote that acquired a life of its own was a thing called shar, the shell

archive, which would bundle up a bunch of files kind of like the way a tar file would except that it was just a shell script so it didn't need to have tar or any networking stuff. If you just had a copy of sh, you could send this one shell script and it would, you would execute the one shell script and it would unpack everything and you didn't need to have any of the other software. And that's been re-implemented a few times and it's one of those things that seems unable to die. But that's the closest thing I can think of.

Hsu: Okay. And of course you, because of your work with BSD, you got to know Bill Joy.

Gosling: Yeah. And Kirk and Eric.

Hsu: Okay, yeah. Kirk [McKusick], yeah. Kirk because, like, and Eric Allman, you mentioned him.

Gosling: Yeah.

Hsu: We spoke earlier about your work on the first Unix version of Emacs.

Gosling: Right.

Hsu: So you had been using Emacs on Multics before that.

Gosling: Correct. Yeah, I was, you know, at Carnegie Mellon there were a few people who were using Emacs on the PDP-10s and that was the MIT version of Emacs that ran on TECO but I pretty much didn't use the PDP-10s. They were slow and, you know, they felt kind of clunky. So I never really got to know Emacs there but I did a contracting job to get Pascal running on Multics and so I did that, but in the process of doing that there was a version of Emacs that was, that ran on Multics that was done by a guy named Bernie Greenberg. And so the Emacs that was written at MIT and kind of like the origin of the name Emacs, Editor Macros, it was a bunch of macros written for the TECO text editor and, and then when then Bernie Greenberg wrote kind of a muscle memory compatible version of Emacs for Multics but he wrote it in MACLISP. So my first exposure was the MACLISP-based thing on Multics. And then when I was done, done with that contract and went back to grad school, you know, and then, you know, back to working on Unix on VAXs and back to using ed and vi and just too sad.

Hsu: <laughs>

Gosling: And I-- There's a theory professor named Mike Shamus and he gave a series of lectures and one of the lectures was on dynamic programming, which is sort of an optimization, a set of optimization algorithms, and in the middle of one of his things I sort of had this epiphany that, oh, you could do a really efficient job of screen, of updating screens with a dynamic programming algorithm. Because one of the problems with dealing with these, you know, glass teletypes was that the baud rate to them was really awful and, you know, if you watched something like vi working on a glass teletype it was doing a lot of painting and a lot of the painting was just, you know, painting the same stuff on top of the stuff that had already been there. And I kind of realized in the middle of this dynamic programming lecture that oh, I could make that a whole lot faster. So I, <laughs> I actually, like, didn't eat for a couple of days and wrote

this screen update algorithm based on this dynamic programming lecture of Mike Shamus's. And then I went, wow, I could really make a killer version of Emacs. So I pretty much stopped working on school for a while. <laughs> And by the end of '78, Emacs was completely self-supporting and you know, the screen painting was really fast and all of that. And but one thing that was a little weird was that there was another student, a guy named Mike Kazar, and he, he came from MIT and was used to using Emacs, but the Emacs that ran on TECO wouldn't run on all of the different operating systems for PDP-10s, but the-- So Mike wrote an Emacs clone in PDP-10 assembler that didn't have, like, all the macro facilities, but it was really, really fast and it ran on all the machines that the Department had rather than just, like, a couple of them. But for some reason or other he didn't like the keystroke bindings that the original Emacs had, so he did his keystroke bindings differently and everybody at, in the CMU CS Department had gotten to know, had gotten used to Mike's bindings. And his editor was called FINE, F-I-N-E, FINE Is Not Emacs.

Hsu: <laughs>

Gosling: And so when I did mine, I did this extension language that was kind of like MACLISP so that you could write all kinds of weird extension packages on it. It wasn't full up MACLISP, so I just called it MOCKLISP.

Hsu: <laughs>

Gosling: And but I copied Mike's key bindings and so when my Emacs started proliferating around the network, around the internet, the MIT folks just flipped out mostly because I changed the key bindings. Of course, you could rebind the keys really easily and there was a key binding file floating around that would map the CMU key order to the MIT key order, but it still flipped them out that the MIT order wasn't the default. But then there was, you know, at one point where I kind of realized that I was either going to be Mr. Emacs for the rest of my life or graduate; I couldn't actually do both, so I decided I wanted to graduate.

Hsu: So then your-- But your display algorithm got incorporated by Richard Stallman into GNU Emacs.

Gosling: Well, actually, it was more than that. He just took all of the source code.

Hsu: Oh, okay.

Gosling: Right. It was, it started out as all of the source code and he just edited the copyright notices.

Hsu: Really? So he just essentially stole your program.

Gosling: Yeah, so--

Hsu: <laughs>

Gosling: Ah--

Hsu: And claimed it as his own? <laughs>

Gosling: Yeah. So, so what had happened was I have this, like, point in my-- in grad school where I realize I'm either Mr. Emacs for life or I graduate. So I decided I wanted to graduate so I kind of went around to all the usual suspects and said, "Is there anybody willing to take over the maintenance of Emacs?" And, and everybody said, "Well, I really love Emacs and everybody here at," you know, MIT or UCLA or whatever, "We all love Emacs, but we actually have day jobs." And they-- And so I couldn't find anybody who would do it. I even asked Stallman and his, his answer wasn't just no, it was more like a frothing hell no. And, you know, mostly because it was for Unix.

Hsu: Mmm.

Gosling: And one of his big things was Unix and using Unix was, back then his attitude was that Unix was the spawn of the devil and he made that really clear. But I did find a couple of guys who were willing to do it but they would need to, like, earn money from it and I had, you know, from the very beginning been very careful about putting copyright notices on Emacs, you know, and the deal that I had with everybody was, you know, write me a letter and I'll send you a mag tape, you know, and the letter was basically agreeing to this license. And the reasons that I had done that was-- So this, this guy Mike Shamus, the guy who, the professor, the theory professor, he'd also gotten a law degree and had gotten involved in a bunch of intellectual property issues. And there was this weird thing that had happened at Carnegie Mellon where Carnegie Mellon was sort of unique in its sort of charter documents in that if a-- At a lot of universities if a student produces some piece of work it's partially owned by the university, so you just, you can't just give stuff away. And they usually have, you know, weird clauses that have evolved and for software around open source stuff that that way you can do it. But back then open source wasn't a thing, but you couldn't give stuff away. But at, but Carnegie Mellon was sort of unique in that it had these rules that said if a student produced some work then the student owned it, which was really strange. And but there had been this one case where Carnegie Mellon has, it's kind of a small university, but it's got sort of two world class departments. One is the School of Computer Science and the other is Performing Arts. And I mean, the Performing Arts Department there is just astonishing. But one of the students there, it was like their master's project or something, they wrote a rock opera and I think it was "Godspell," not exactly sure. But then that, but it turned into a big financial blockbuster and the university tried to retroactively change their rules about intellectual property and that turned into a big saga. And, and one of the things that came out of that was this sort of lingering apprehensiveness about intellectual property rights. And in the Computer Science Department, there was another student who had built a piece of software that had gotten really popular on the internet and it was a text formatter called Scribe. It was done by Brian Reed. And he also got into a bunch of, you know, the university tried to pull stunts with him. And so after that history, I decided that I should be careful and I talked to Mike Shamus about it and he said, "You know, do, you know, put this header on all your source files and make sure you get a letter back from folks that basically says that they acknowledge the dut-dut-dut-dut" and so I was really careful about that. But then when I had decided, you know, I wanted to graduate rather than being Mr. Emacs, I found these two guys who ran this little company called Unipress, it was literally two guys in a garage, and I said, "Look, this needs to be free for universities and not ridiculous for everybody else." And they said, you know, "Fine. We're just two guys in a garage. We don't need much. And this seems to be

awfully popular, so we think we can actually, you know, pay our rent and feed ourselves." So that was fine and they were doing really well. And then Stallman freaks and he gets a copy of my source code, does a whole lot of editing. He doesn't actually-- You know, he edits, like, almost all of the copyright headers, but he doesn't edit all of them and he only kind of thinly edits it and then he re-releases it as GNU Emacs. And then IBM and Digital Equipment pick that up and start distributing it. So these, so the two guys in the garage who had been doing okay suddenly find that IBM and DEC are distributing their thing for free and they're dying. So they decided to sue DEC and IBM and they got a, you know, and that turned into a big, big court case. They won. It was kind of a pyrrhic victory. You know, so IBM and DEC paid them some damages, but that didn't stop GNU Emacs. You know, they didn't get GNU Emacs pulled, so it just sort of carried on and these guys sort of shifted their business, which always made me feel kind of bad for them because they really got shafted. And, yeah, so and at the very beginning of GNU Emacs it was literally line for line, I mean, they actually, you know, there were expert witnesses who, you know, by eyeball, you know, searched through the source code and went, "Yeah, it's the same." They actually found that some of the early GNU Emacs source files that were being distributed, they hadn't, he hadn't actually changed all of the copyright notices. So it was like, duh. But then it sort of took on its own life and it's become the GNU Emacs that everybody uses.

Weber: But Stallman was not involved. No one went after him because he had no money, right?

Gosling: Right. I mean, you can't sue a homeless person, right, which is, you know, he-- Yeah, he had sort of weird views on, you know, economic models at the time.

Weber: And personal hygiene.

Gosling: Yeah. Yeah. And, you know, I realize I'm a radical. I like to actually have a bed to sleep in and food and I will occasionally sleep in my office, but not as a regular thing. And I haven't done that for a long time.

Hsu: Earlier you mentioned you had taken on that contract job on working on the Pascal compiler. Could you talk a little bit more about that?

Gosling: Yeah. So that Pascal compiler was a lot of fun. It actually had kind of a long and twisted history long ago and far away when Niklaus Wirth and company did the first Pascal compiler at ETH, it was always distributed in source form and it had, like, no license or anything on it. And it had showed up at the University of Calgary and people started teaching courses using it both as a-- as an example compiler and as a compiler to use for courses. And, and a few of us started tinkering with the compiler and, you know, changing Pascal in some interesting ways. And one of the guys actually turned all that work into his master's thesis. And it was a compiler that generated code for the CDC 6000 and 7000 series machines. And Honeywell had this, they needed a Pascal compiler for Multics and this was at a time before they had completely killed Multics. As the more I got to know about the way they were treating Multics and what they were doing with it, it's like, you guys, of course Multics died; you were idiots. And, and so I got hired to produce a Pascal compiler for them but I had, like, three months to do it and the plan was that I would take this sort of souped up Pascal compiler that I had worked on at the University of Calgary and have it,

you know, convert it to generate Multics code, so I did that. And, you know, getting to plumb that into the lowest reaches of Multics so that it would-- Multics is a fascinating operating system, but their base implementation language is PL/I and the base PL/I compiler generated atrocious code. So one of the things that I got into doing was a lot of, like, benchmarking between Pascal and PL/I and it's really easy to beat PL/I. And but, you know, there's a lot of complexity in dealing with the addressing system on the Multics machines because it had these 72 bit pointers that had a lot of, like, authentication information and it was, it's a pretty cool system. But yeah, I ported this thing over and as a result of that I'm now listed on multicians.org.

Hsu: <laughs>

Gosling: One of the things that was sort of weird about it was that-- I don't really understand the org structure at all, because there's Honeywell and Honeywell Bull, which was the French company, and they're kind of joined or they were kind of joined but they were kind of separate. And, and so Honeywell Bull did a different Pascal compiler that had, like, a big team of people working a long time and then Honeywell USA had hired me to do kind of the same thing. And I don't know when the French one started, but, you know, I sort of discovered at some point that I was actually in a horse race and, you know, that just turned into yet another lesson in politics, but.

Hsu: You also worked on another compiler for a language called Mumble.

Gosling: Oh, Mumble, the Most Unlikely Micro Assembler.

Hsu: <laughs>

Gosling: Mumble was a project that I really liked. It was actually the project that I did as a grad student at CMU that convinced them to make me an RA rather than just a regular student, just as the one and only tuition paying student. And so there was this-- There were these machines that were built that were these horizontally, very horizontal microcoded bit-sliced machines that were built at CMU that were the memory controllers for a machine called CM STAR. And the instruction words were bizarre because it wasn't like it was an instruction that executes; it was a series of bit fields and it was quite a long instruction word. It was, like, somewhere around 80 bits and it was bunches of fields that controlled different sets of gates. And so you had to think of all of these bit fields as really as separate instructions. And so, the whole thing was this kind of, like, data flow machine, and so if you wanted to add A and B and store it in C, in one instruction you'd have to pull A out of a register and put it on to that bus and then you'd have to put that into, you know, the, in the next instruction, you would have to take that and add it and then later you would have to store it. So, you know, in, like, three or four machine cycles, you know, you could compute A equals B plus C. But, but each of those phases of A equals B plus C would be in different words, but they would be different fields in different words and this thing was so chaotic that all the code for it was written by hand. And the architecture had ended up this way partly because it was easy to build because there were these bit sliced processors that you could buy that were cheap and easy to put together and you would end up with something kind of like this. But also because the thing was just awesomely fast. I mean, there was no instruction decoding. I mean, literally, you know, it would

take, you know, 3 bits from here and gate it into that multiplexer. And so, so if you looked at the microcode that people were writing for this, it didn't actually use the instructions very efficiently and I'm not exactly sure how I got roped into this but it was just this, like, fascinating thing and I had these weird ideas about how you could view it as not, like, a big wide instruction, a very wide instruction, but a whole lot of little instructions with a dependency graph in between them and, and the dependency graph could actually have, like, residency constraints, like this instruction couldn't be in the same word. This instruction had to be, like, one cycle later or two cycles later. And you'd build this, like, huge dependency graph out of all these little code fragments and then go through this, you know, topological sort effectively of all these things to do a, you know, solve the dependency graph to compress it out. And it got really complicated when there were, like, conditionals and loops. And, you know, I kind of got out of control building this thing because I thought it would make a good Ph.D. thesis topic and I actually think it would have made a really great Ph.D. thesis topic. But there's sort of this history at CS at the time of you know, sort of systems thesis, you know, things that were about software not really working as thesis topics, because most of the faculty was, like, theory folks. And so I got kind of, like, dissuaded from actually trying to turn that into a thesis, but I built this compiler. And one of the things that was interesting about it was that there was, you know, there was, like, the group of people who were writing microcode for this because this was actually a set of memory controllers for this much larger distributed multiprocessor and it controlled the message bus that, you know, it was sort of the logic for routing messages between these things. And there was this one group who was building that microcode, you know, just like writing microcode. And I did this compiler and one of the, there was this one guy who said, "Gee, I'll bet I could use that to write the microcode." So he and I kind of worked together. He wrote the OS piece, I wrote the compiler piece and his worked really, really well. And because you, you know, because you could write vaguely arbitrary code to run on these memory controllers, you know, he could put more of the OS into the memory controller, so he actually wrote very little code that ran on the edge processors. He was able to, you know, for his little OS he could put most of the OS actually in the memory controller and his performance numbers were really nice and besides, it was like one of him and three other guys versus the other group of, like, three or four people. So it was like one guy versus this team of three or four guys and he kind of was able to, like, run circles around them. And the performance of this thing was just, like, amazing and the fact that he could put a lot more code into the infrastructure, it was pretty cool. But one of the things that we kind of figured out was that the fundamental architecture of this multiprocessor was goofy because you could actually, you know, because the routing bus controllers were generally programmable and really, really fast, if you just took all of the processors, all of the edge processors, the ones that are meant for doing real work, and you threw them in the trash and you just ran your app, you know, in the switching fabric rather than in the processors, it would be faster. And one of the people who was involved in that was also heavily involved at Juniper Networks and for a while he was trying to get me to go work at Juniper and I was, like, yeah.

Weber: Who was that?

Gosling: Pradeep Sindhu. Yeah. And, you know, part of me was really, really tempted and part of me was like, I don't want to be that deep and low in the plumbing, but.

Hsu: So you also worked on an intelligent mail handling system for Herminet as a summer intern.

Gosling: Yeah. Well, it was, I don't know whether to call it a summer intern or a contracting position. It actually stretched over a couple of years and I spent a bunch of time in San Francisco working on that project. And this is a really long story. I'm not going to get it done, but you'll have to ask me the question again. But as sort of background, the guy who sort of started this project and ran it was this guy named Fernando Flores and I don't know if you've heard of him. He wrote some books with Terry Winograd.

Hsu: Yeah, yeah, yeah. That one.

Gosling: But he's Chilean and one of these sort of freak of nature smart people. And he was the Economics Minister for a while and the Foreign Affairs Minister for Chile under Salvador Allende. And when, you know, the big U.S.-backed coup with Pinochet happened, Fernando was one of the people who was captured and thrown in prison and just barely escaped with his life.

Hsu: He'd worked on Cybersyn?

Gosling: Yes.

Hsu: Wow.

Weber: He was a computer guy then, okay.

Gosling: Well, he wasn't exactly a computer guy, but he was sort of a theory of organizations guy and how, you know. So he thought a lot about how societies and governments operate and so it was this whole-- that whole Cybersyn thing that he tried building. He was the guy who organized all the people who came together to put that. He was, like, the minister in charge, but it was his thing. And, and then the whole Pinochet thing happens and then Terry Winograd, who was a professor at Stanford, he was involved in the Amnesty International team that worked heavily on-- No, not Amnesty International-- or maybe it was. Who's the, there's one of these that does massive letter writing campaigns.

Weber: Yeah.

Gosling: Maybe it is Amnesty International. Anyway, they got him out. He came to Berkeley. He ended up getting a Ph.D. at I think Berkeley and then being a faculty member at Stanford or the other order, but somehow or other he got involved with Werner Erhard, who was kind of a nut case. So he got--

Hsu: The est?

Gosling: Yeah, yeah.

Hsu: Yeah.

Gosling: Erhard Sensitivity Training.

Weber: Seminar Training.

Gosling: Yeah.

Weber: Isn't it?

Gosling: Sensitivity. Anyway, Fernando got Erhard to bankroll this thing and it started out, so I'm not exactly sure, but Fernando was looking for somebody to build this prototype for him that came out of some of the ideas from this Cybersyn idea and through an odd path, he knew one of my classmates at Carnegie Mellon and my classmate at Carnegie Mellon said, "James, this sound sounds like something you could do in your sleep." And so I started working with Fernando and he set me up with an apartment in San Francisco and Werner Erhard would come to visit every now and then and he was a uniquely weird person. He's--

Weber: Describe.

Gosling: Huh? Well, he, I mean, besides, you know, the obvious sort of leader of a weird California cult thing, he was a really serious germaphobe and neat freak. And because he had more money than God and he was a charity, so he didn't have to pay any pesky taxes, he had, like, a front team that would go ahead of him and clean things. And so if he was coming to visit me, this group of four or five people would come to clean my apartment but, you know, at kind of like a, you know, microbe level. You know, they would have, you know, toothbrushes with stuff on, like, all the light switches and they would, you know, wipe down all the walls and sanitize all the furniture and, you know, very-- You know, like, after they were done cleaning they would go to, like, the furniture and all the cushions, they would spread out all the fabric to be flat and smooth, you know. So all the furniture would look more like a CAD model than a piece of furniture. And you know, they would line up all my cables, because this was, like, both my apartment and the lab and I'm sure that there were many regulations ignored, but, yeah, they would want to, like, rerun cables and things and it was just, <laughs> it was just always odd. But, yeah, so, I mean, the follow-on story of that mail system is fascinating and long. I could spend hours talking about that. And it's been on my to do list forever to rebuild it, because Fernando had some great ideas and it was significantly ahead of the state of the art at the time. You know, and so I could build a system that would do all of it, but because the networks really weren't there, it was kind of hard to do in the way that he really wanted. But these days, the networks are there and the big issue is how can you do some of these things and still fit in with the mail standards and I think I kind of know how to do it, but because every now and then I think about it and, like, how to do some of these things and it would be a radically weird mail system that could be pretty cool. But, yeah, for another day.

END OF THE INTERVIEW

