

# Oral History of Anders Hejlsberg

Interviewed by **Becky Monk** for the Microsoft Alumni Network

August 7, 2024

## Preface

The following oral history is the result of a recorded interview with Anders Hejlsberg as conducted by Becky Monk on August 7, 2024, at Microsoft Studios in Redmond, Washington. This interview is part of the Microsoft Alumni Network's Microsoft Alumni Voices initiative. The goal of this project is to record the institutional history of Microsoft through the recollections of its former employees, so that the information may inform and inspire future generations.

Readers are asked to bear in mind that they are reading a transcript of the spoken word captured through video rather than written prose. The content reflects the recollections of the interviewee. The following transcript was edited by the Microsoft Alumni Network, which holds the copyright to this work.

## Interview

Becky Monk: All right. Well, we're going to get the hardest part out of the way first. Just kind of a 30 second who you are, when you started at Microsoft and what you're doing.

Anders Hejlsberg: Okay. Hi, I'm Anders Hejlsberg. I've been on Microsoft since 1996. I'm a technical fellow in the developer division and I work on programming languages and software development tools. And I have pretty much done that for my entire career, which now spans a little over 40 years.

Becky Monk: Wow, 40 years?

Anders Hejlsberg: Yeah.

Becky Monk: Okay. Let's go back to where you were born and grew up.

Anders Hejlsberg: So I was born in a suburb of Copenhagen in Denmark just north of Copenhagen and grew up there. It was a very nice middle-class safe upbringing. I have a brother. He and I had a lot of fun just riding our bikes around. We were sort of out in... There was nature close by and so we were fairly self-sufficient. Of course, back then you didn't have phones or anything and you would just leave a note to your mom that, "We're there and there and we'll be home at 5:00." Or she would leave a note for you when you came home from school. And in fact, I have a stack of those old notes from back then.

I went to school at a school called Holte Gymnasium. And went there ever since the middle of second grade all the way through high school. And that school, I was fortunate enough, was one of the first schools in Denmark to offer the students access to a computer, which I think must have been back in for me probably '76, '77, the first time we got to lay our hands on an actual computer and we got to program it. And I often joked that for the first two weeks the teachers would teach us how the computer worked and then for the remaining time we would teach them how it worked. Because we just got in there and tried everything, programmed it in machine code, wrote bootloaders, tortured the hard disk drive and the paper puncher. We wrote programs that

would do ticker tapes and all sorts of stuff. We knew that thing inside out.

Becky Monk: What was it about that first computer that really sparked your imagination? Because I can see the excitement even thinking about that first computer.

Anders Hejlsberg: I don't really know why. I think there's something very appealing about the little world you have in a computer where you get to control everything. And there's something very satisfying about implementing something complicated and then seeing it work or understanding why it didn't work and then figuring out how to make it work. I don't know, it's very addictive, at least to me it is. I know I realize to a lot of people it's not at all, but I am somehow of that kind.

Becky Monk: Well, it sparked your imagination and it sparked a career path. How did you decide that, that was the career path you wanted to take?

Anders Hejlsberg: I was always interested in technical things, cars and anything having to do with electricity. And my dad was an engineer and so I was pretty certain that, that was the route I wanted to go and I kind of knew that even in high school. And so, when I graduated high school, I applied and got accepted at the Danish Technical University, which is a big polytechnic school, which is also pretty close to where I grew up. And started there in '79, the year that I graduated high school also.

This is before... They didn't actually have computer science as a line at the time. You could do that at the Copenhagen University, and that was back in the mainframe days and whatever. But I knew that I was not cut out for the sort of lecture hall experience. I liked more of a classroom setting that they had at the Polytechnic. And so I went there, but to get an electrical engineering degree basically. Which of course with as much programming in it as possible. And let's see, for the first year students, they do a trip with seniors that take you to some, I think we went to some cottages or something or some camp somewhere.

And I like playing cards. There's a particular card game called Mausel that we would play, which no one knows around here. But it's a game for money. And so we were playing and I forget whether I was the one losing or someone else was losing to me, but there was a guy named Preben Madsen that I got to know through this card game and he was talking about this computer store that they were computer company that they were just founding. And I kind of got interested in it because I was very interested in...

This is right when 8-bit kit computers were starting to happen and I was definitely wanting to get one for myself. And they were selling those amongst other things. And so, I ended up getting involved in that company, a company called PolyData. And that happened pretty early on. Basically, right after I started my studies within a year I was sort of engaged in that and became... Then the company restructured and I became one of the co-founders of that branch of the restructure, and that was then ongoing whilst I was studying.

Becky Monk: So with this company, was that where you were really getting into writing languages?

Anders Hejlsberg: Well, with that thing, it was wonderful because it gave you a very, very deep introduction to exactly how the kinds of computers that we all use today worked. Literally, from here's a printed circuit board and all the chips and you got to solder it in together and make it work and figure out why it doesn't work with an oscilloscope and whatever. So there was the hardware side of it that you learned a lot about, but then of course there was also writing software for these things because then once you did get it to work and you turn it on, then it just sat there and did nothing.

And so, now you got to make it do something. And so what can you do on it and how do you program it and what are the different ways you could program it? These machines back then typically all came with Microsoft's ROM BASIC in them, and that included the one... The one in particular that I was interested in was a British kit computer, a Z80-based kit computer called Nascom. And it had Microsoft's 8K ROM BASIC in a ROM on the motherboard.

And of course you start out writing in BASIC, but BASIC, that's not for real programmers. It's too slow, it lacks stuff but it's not close enough to the metal. And so, I pretty quickly got interested in how do you write really efficient code on these machines? And that meant writing in assembly code or in machine code. And so, I started writing a bunch of different software after I'd written games and BASIC and sort of gotten tired of that, started writing assemblers and disassemblers and on-screen editors and wrote a

small operating system. Disk drives, floppy disk drives were just becoming available. Before that it was like cassette tape.

But I also got interested in this programming language called Pascal. Because at my high school we had been taught ALGOL because that was what was on that... We had an HP 2100 at that high school with 32 KB of ferrite core memory. And, oh my God, that was state of the art. But you could program it in ALGOL. And I liked that language and I was like, "Oh, I should see if I can write an ALGOL compiler." And then my friend was, "Oh, there's this new thing called Pascal. It's done by the same guy, but it's much better." And I was like, "Oh, okay." So Pascal, it was.

And then I wrote a small compiler for the Pascal language after first teaching myself how to write compilers. And it was all written in assembly code and machine code because that was the only way you could squeeze it on a machine that, at most, could have 48k of RAM because some of the address base was taken up by Microsoft ROM BASIC and other things. And eventually that ended up being a 12k Pascal compiler and runtime and on-screen editor.

And we even built a little board where you could put three e-prompts on and the Microsoft ROM BASIC, and then you could slot it into the ROM with the BASIC with a big switch that made the machine either boot up in BASIC or in our Pascal little system. And when you boot it up and that you basically had this, it's the earliest predecessor of Turbo Pascal effectively, but a very small subset of the language but that interactive feel of run and compile run immediately.

Becky Monk: So just so everybody understands, can you explain kind of the significance of what that compiler was doing and why that mattered to computing at the time?

Anders Hejlsberg: Sure.

Becky Monk: Because I think we all think it happens naturally now.

Anders Hejlsberg: Well, even to this day, programming languages tend to fall into two categories. Either they're dynamic interpretive programming languages or they are languages that are first processed by a compiler into binary code that then is run by the CPU. Now, the Microsoft ROM BASIC I was talking about, that's the most probably prototypical and archetypical example of an interpreted language where you type in the code and then when you say run, then there's a program that runs your program. It interprets your program, it interprets the text that you put in, and then eventually that turns into actions and variables being set to values and so forth.

But there's an awful lot of overhead associated with that. All of the overhead of doing the interpretation is effectively cost that you pay at runtime and therefore your programs run slower. And so, if you really want maximum throughput out of a machine, you need a compiler. And a compiler processes your program and turns it into machine code and finds errors in the program if there are errors. But then once you have the machine code, that's what you run and that stuff runs at native speed much, much faster.

Now, the interactive systems, the interpreted systems were always very interactive. Like in Microsoft ROM BASIC, 10 print Hello, World. 20 GOTO 10 and run and then off you are running. Immediate feedback. And it was fun and interactive. With compilers, it was much more like, "Well, let me first fire up my text editor and type in my code." And of course, no one tells you whether this is going to work or not. Then you save that to a file. Then typically on those old machines, because there was very little capacity, then you'd take out the floppy disk with your editor, and then you stick in the floppy disk with your compiler. Then you'd tell the compiler to please compile your program. Then it'd go ka-chunk ka-chunk. It would load the compiler, and then it'd do multiple passes over the source code and build maybe intermediate code, and then finally write out an object file.

Then you have to use a linker to turn that into an executable. And then finally, you could try to run your program and then your program would blow up. And then you'd go, "What just happened?" Because you're not in an interpreted environment anymore. You're running raw machine code, and if it goes wrong, maybe all of a sudden your screen turns into an amusement park, just blinking characters everywhere, and you have no idea. It's like the equivalent of the blue screen.

And now you got to figure out, "well, what's wrong with this?" So now maybe you need a debugger. And so there's this whole workflow that was incredibly arduous and it could take you minutes, if not hours, to get from, "I typed in my program," to it finally runs. And then in order to fix something, you go through that whole

cycle again. And that to me, it just felt like this is incredibly clunky, surely we can do better, and so...

Becky Monk: And so better was Turbo Pascal?

Anders Hejlsberg: Well, so Turbo Pascal was... Well actually that little 12k Pascal compiler that I talked about for that 8-bit kit computer was really the predecessor of Turbo Pascal. And the idea was there that you would keep it all in memory. You would have the compiler, the runtime, the editor, it would all be in memory. Your source code would be in memory. When you compiled the machine code would go straight into memory, and then when you ran, you would run the code you just generated into memory.

So there was no intermediate medium that you had to first save it at and then take out and load another program or whatever. It was all there. Now, in order for that to be possible, then clearly this compiler could not consume a lot of memory because you had to have everything in memory. And hence, I had to write this all in machine code, and that was the case. And that was then what grew into Turbo Pascal, where the full compiler, editor, runtime took up only about 32k of memory, leaving enough room over for you to have your program in memory and then be able to compile it in memory and run it in memory without ever having to save anything on a disk drive. And that gave you that instantaneous feel of, "Oh my God, this is just an interpreted language except it runs 10 times faster," because you're running native code. And so it was sort of like the best of both worlds.

Becky Monk: So that allowed for kind of a revolution in the industry really. Can you talk a little bit about what was revolutionary and what the change was to the industry? What could people do?

Anders Hejlsberg: Well, I think with Turbo Pascal, which this product grew into, and now we're talking... The first version of Turbo Pascal was released in '83, at which point everyone was on 8-bit machines typically. The PC was just launching, the first 16-bit machine that became a popular... And this notion that all of a sudden you could have a full-on compiled programming language and you could use it in an interactive manner had never really been done before. It was always like this choice of, "Oh, I can either have easy and pleasant to work with in an interpreted, or I can have clunky and unpleasant with compilers and linkers and debuggers and on-screen editors or text editors or what have you."

And I mean, the competition at, the Pascal compilers existed. They usually cost 500 bucks. They were slow. They did not have an interactive development environment at all. It was very one program at a time. And with Turbo Pascal and Borland, I think two things really rocked the industry. One was that we built the world's first integrated development environment that truly gave you that feel that development environments have today, all of them.

And the second was that we dropped the price by 10x from 500 bucks to 49.95. And the combination of those two was just explosive once the product was launched. Typically, the reaction people would have with the product is, "Well, that can't be true," or, "That must be a scam." Or, "That's simply not possible." Do you know what I mean? Until they then got their hands on it and

realized, "Oh my God, this thing actually does what it says it does. Not only that, it does it better than the competition for a 10th of the price." And it became a runaway success very quickly.

Becky Monk: What could people do? Was this mainly for developers? Was it for the general public? Now that it's \$49, who was the main user of this?

Anders Hejlsberg: I think the hobbyist. You got to remember at the time, people who bought computers were... Well, in the beginning with the 8-bit, put them together yourself micros it was all hobbyists. And then it grew and grew. And it wasn't really until, I'm going to say VisiCalc and the Apple II, that for the first time, there was a piece of software that would make you buy the hardware. You didn't actually care about the hardware. You cared about running that program. Do you know what I mean? But before that, it was really, you cared about the hardware and you were a geek, if you will.

And of course you liked to program this thing because what else could you do with it? That's what you did with computers. Well, you might buy some piece of software that someone else had written, but generally, people at the time wanted to learn how to program their computers. And of course, there was a huge educational push also because people saw what was happening. And PCs were starting to make their way into schools, and you needed good teaching languages. And it turns out Pascal was a language that was created specifically to be a pedagogical teaching language. And so, it became very popular in that way too.

It came from the strangest places. We used to joke, the entire Eastern Bloc, it turns out, was raised on Turbo Pascal. We had this joke that we have a site license in Russia, we are in the Soviet Union. We sold one copy, and then they just freely distributed everywhere because there was no copy protection on this. Because our reasoning was, "Hey, for 49.95 it's not even worth it to rip it off because for 49.95 you also get the manuals." And they were actually good and they taught you how to work.

But to this day, I meet people who talk about the... Turbo Pascal made its appearance in '83, and then the Soviet Union fell in '89, and then a whole bunch of programmers who had been employed by the state all of a sudden had nothing to do. And I meet people who tell me to this day that, "Turbo Pascal saved my life, literally. It gave me something that I could do. I could program using that, write software, sell it to other people and feed my family." I'm just like, "Wow. That's mind-blowing, right?"

Becky Monk: Did you have any idea that...

Anders Hejlsberg: No, I was just some dude from Denmark who was interested in computers and thought, "Hey, this was a good way to do it."

Becky Monk: That's incredible. Okay, so everybody is all over Turbo Pascal. They know you're the guy. How do you end up at Microsoft?

Anders Hejlsberg: Well, that's a longer story there too. So the small software company that I got involved with, or the guy that I met in my engineering studies, we had a company called PolyData. And we had a computer store, the first computer store in Copenhagen where you could walk in and buy, at the time, kit computers. Through our connections, we met some other guys who had another small computer company in Denmark, and they wanted to get into the software business.

In fact, they had already founded a company called Borland, and they had two products. One called I think Menu Master and another one called Word Index, which was a menuing system for C/PM. And Word Index was a table of contents generator for WordStar, popular word processor at the time. And they had written it in Pascal, but another Pascal, MT+, I think it was called. A terrible Pascal compiler.

Anders Hejlsberg: And we met them at some point, my partner met him and goes, "Oh, we have something that's much better. You should check out this thing." We were selling the predecessor of Turbo Pascal at the time. And he gave them an evaluation copy of that. And then they came back a week later and go, "Wow, this is amazing. We should make a deal. We should build a product together where we put a WordStar compatible editor on top of this, and change the runtime library, and sell it in the U.S. And so we built a license deal, or a royalty deal. And then we got together and put a different onscreen editor on it and rebranded it Turbo Pascal, and then started selling it in the US.

In fact, I remember I was always under the assumption with this agreement that we had, it was just a royalty of net sales. And I was always assuming, "Oh, of course they're going to sell for around

500 bucks, and so we'll make so-and-so much per copy." And then at one point they'd come back and go, "We're going to sell it for 49.95." I go, "Are you nuts? That's crazy. I mean, we'll never make any money this way." Of course they were right, because we sold orders of magnitude more copies. But that's how I got involved with Borland. And in the beginning it was really just a royalty deal between Borland and my company. But eventually that became all I did. And I became a Borland employee, and traded the deal for shares of Borland. And then eventually moved to the U.S. in '87 after commuting back and forth from Denmark, which it was a royal pain in the neck back then, because there was no internet. There was no nothing.

In fact, even telecom, it was controlled by the state-owned telephone companies. And the best modem you could get in the beginning was a 1200 baud modem, which that's 120 bytes per second. You try to transmit anything across the Atlantic on a poor quality phone line with crappy transmission software that if you get halfway through and then all of a sudden the line drops, then you got to start all over again. It was terrible. We would DHL floppy disks back and forth. That was actually quicker than try to transmit it electronically in the beginning.

And so of course, by '87 it had gotten better, but still there was no internet, and it was bad. So I moved to the U.S. to be closer to the action. And then with Borland in the U.S. from '87 until '96. And built a long succession of versions of Turbo Pascal. I think we got up to Turbo Pascal version 6 or 7. And then we built the successor to Turbo Pascal called Delphi.

And now we're in the early nineties, early- to mid-nineties, and this is when graphical user interfaces were starting to happen. And Microsoft had launched this really revolutionary product called Visual Basic at the time. And it was one of the first ways that took the integrated development environment into RAD, or Rapid Application Development, where you could graphically visually build your dialogues and your forms alongside with the code. But again, it was an interpreted language, it didn't have object orientation, which was the thing of the future. And it didn't have native code. It didn't have a whole bunch of things. And so we felt that, hey, we have great technology and great know-how about how to build compilers that are really fast, we need to learn about how to build a graphical user interface, and we need to learn how to build an extensible object model that allows users to write new visual controls and whatever. And we set the work doing that. And it took several years.

And then we added an extra year onto it in order to brand it a client server product, because that was how you got into the enterprise at the time. Otherwise, you would still just try to sell the hobbyists. But oh, client-server, products like PowerBuilder and whatever, that was like where the money was at. And so in '95 we launched Delphi. And that actually became a very successful product. Also, to this day, lots of people are programming Delphi, and lots of apps out there are written in Delphi. But now we're in '95, and that's when Microsoft started. Ostensibly, that's when Bill got really irritated. It's like, "Why is it that they keep beating us in software development tools?" And Microsoft had at the time, Borland had gone through some ups and downs, particularly in the early nineties. And when companies go through ups and downs, that means in the downs they let go of people, and then these people go spread themselves out over the industry.

And so there were a lot of Borland people at Microsoft, including, say, Brad Silverberg, for example, who had been VP of R&D at Borland. And he would call every summer, "Yeah, I wasn't ready. I wasn't ready." Then in '95 I was sort of, "Okay, maybe I'll come up." And of course, it was always in late summer, early fall, when the weather's beautiful, you come up to Seattle, oh, my God, this is so green and wonderful. And they put you up at the Edgewater. It was wonderful, right? But I still decided, in '95, I decided I still want to do one more go round and build a 32-bit version of Delphi. And so next summer, in '96, I finally decided, okay, it was time.

Becky Monk: What was that call from Brad Silverberg that made you decide now was it?

Anders Hejlsberg: It was a combination of many things. I think it was the incredible opportunity to work, because the writing was on the wall. Microsoft was going to be a juggernaut, and it was way bigger at the time than Borland. And to have a company, to get in on that, would be incredibly exciting. And at the same time, Borland was going through yet another round of layoffs, and yet another round of confusion about who the company wanted to be. In a sense, Borland's expertise was always software development tooling, but the company always wanted to be something other than that. They wanted to sell business software, a word processor, spreadsheets, whatever. And so, if you were on the development tool side, you always kind of felt neglected, do you know what I mean? And not enough attention was being paid to that part of the business. And that was the one thing about Microsoft. They took their development tools very seriously. And it goes way back to Bill, having built Microsoft BASIC, and was big into programming languages. So that always had a high star at Microsoft, so that made it attractive too, I think.

Becky Monk: So Brad brings you in. Were you working directly for Brad then when you started?

Anders Hejlsberg: When I started, my recruiting was, it was Bob Muglia and Brad Silverberg that I talked about. I remember interviewing with Paul Moritz. And when I first started, I'm trying to think who, I ended up reporting to Yuval Neeman, I think, who was running DevTools at the time. And my immediate manager was Brad Lovering, who was in charge of, believe it or not, Visual Basic 6.0. But I was brought on board at the time ostensibly to be the architect of Microsoft's Java development tool. Because Java was this thing that, I think it surfaced in '96. And this was sort of commensurate with the internet, all of a sudden, taking over the world. And, oh, my God, here was a language made for the internet. At least that's how it was framed. And you could write applets in your webpages, and it was like this compile to bytecode, write ones run everywhere, object-oriented.

It checked all the boxes. And we were all like, "Oh, God, we're dead. There's never going to be another programming language. I guess we got to go work on this thing." And then, I mean, there was a fraction inside Microsoft. I remember when I came here that really believed that, yes, we should write the operating system in Java. I mean, that was how crazy it was. There was the Java Fund, which was this VC company whose sole purpose was to fund other companies writing software in Java. It didn't matter what the software was, as long as it was Java, fine, you can have money. That's how crazy it was. And so I came to work on Java, because that's the only thing you could work on at the time. And was part of building.

Well, right when I came in, it was I think Visual J++ 1.0, which was really just sort of like a product where we took the C++ compiler out and put in a Java compiler instead. But then we set off to build a Visual J++ 6.0, which was going to be the first rapid application integrated development environment for Java. And that was a lot of fun to work on. And we built a really cool product. Except then the whole lawsuit between Sun and Microsoft happened. And our product got enjoined literally by a judge in San Jose that forced us to put up warning dialogues, "You were about to use Microsoft proprietary extensions. Do you wish to proceed?" "Yes/No."

Because we had added some things to the runtime, because hey, if you're going to write on Windows and you're going to write in Java, then obviously you want to be able to exploit the underlying platform. But of course, Java was all about right ones run everywhere, and it was all about sort of a least common denominator experience, even with your UI. It was always funny, these official Java apps, they always looked a little funny, because they weren't actually natively exploiting the platform, and they didn't look native, they looked different, and didn't work as well. And so we wanted to give our customers the best possible experience, but boy, that did not work out well.

Becky Monk: But they came to you and said, "We want something different." How did you end up with C#?

Anders Hejlsberg: Like I said, what happened was there was this really strong push inside Microsoft to make Java a first-class supported programming

language and even start using it in our operating systems and in our apps and whatever. And then the lawsuit between Sun and Microsoft happened. And that relationship completely went sideways. And at that point, it became clear to ourselves and the leadership that it probably wasn't the wisest decision to base your core business on a product that you licensed from somewhere else. We needed to build something where we could control our own destiny and deliver the best possible solution to our customers. And that was, I think, the genesis of .NET. This idea of building a runtime that could compete with Java, but at the same time offer seamless interoperability with all the technologies that already existed in Windows, like COM, and OLE, and what have you. The way that apps were built to interoperate before we had the .NET runtime.

And of course, we needed a programming language on this platform. And one of them was going to be, we were going to build, we called it the Common Language Runtime, or the CLR, because we knew from day one we wanted to support the ability to run multiple different programming languages on the same platform. And one of these programming languages, of course, was going to be Visual Basic. And the other was going to be C++. But we also full well knew that what people really wanted was the ease of use of Visual Basic with the power and expressiveness of C++ in a sense. Because we saw people write apps in Visual Basic, and then once they were happy with the prototype they wrote in Visual Basic, then they translated it into C++, which meant you wrote the app twice, and you had two teams, and it was a very costly the way of doing it. What if we could create a language that really could span that spectrum? And that was what motivated us with C#. And that was the marching orders, if you will. We need a language like that.

Becky Monk: How did that conversation go? Who was in the room? Did they call you in and say, "Anders"?

Anders Hejlsberg: Yes and no. I mean, we were building the .NET runtime, but it was before it was named .NET it had many different names. It was COM+, it was NGWS, at one point it's Next Generation Windows Services and whatever. The .NET name didn't really arrive until a month before the PDC where we actually released it. But we were building this COM+ runtime that could run bytecodes, that could be retargeted, that could support multiple different languages, that had garbage collection and exception handling built in. And of course, it was possible to put the Java programming language on top of that, but that probably wasn't a good idea, right? I mean, because lawsuit. And then there was this customer demand for something that was like Basic, but with the power of C++.

Becky Monk: Who was in the room?

Anders Hejlsberg: I think at the time, developer tools were run by Paul Gross at this point, who had come from Borland to Microsoft along with myself. Also in the room, when we did the early reviews, and the whole decision... The decision process rolled up to Paul Moritz. I remember Brad Silverberg was not really directly involved in it because he was doing Windows 95, and he was over on the OS side of the house. I think Bob Muglia was involved as well in making some of these decisions. And I was lucky enough to be given the task of building this language.

I remember writing a four-page white paper called What Is COOL, because that was the name of the little prototype language that we were in the process of building. It stood for C-like Object-Oriented Language. And we had all our file extensions were .cool. It was very cool. And it basically laid out a blueprint for this language that would give you the power of C++ with the ease of use of Visual Basic, built on top of the .NET runtime. They yet to be named .NET runtime. And well, I guess people bought division. And so I was given the marching orders.

Becky Monk: How big was your team to do this? Was it you and...

Anders Hejlsberg: It was maybe 10 people, I would say. At the time I was working with Scott Wiltamuth, became the product manager. And we pretty quickly formed a design team of about six to seven people. It was sort of rotated. Some people would leave, and then we'd find replacements, put them in. But from the get-go, we had two design meetings a week, each of two hours. We took copious notes of all... No, we had three design meetings, sorry. Monday, Wednesday, Friday. We took copious notes, and basically built a spec for the language, whilst in parallel implementing the first compiler for the language. The compiler team was headed by Peter Golde. And we wrote that first compiler in C++. We didn't use all the features of C++. But because we didn't have the language, we couldn't write it in itself. So we weren't bootstrapping yet. So we were writing the compiler in C++ as we were evolving the spec.

And that probably took from get-go until the first workable implementation, 12 months maybe, 18 months. We had the first working language, where you could actually write apps, and there

was a language spec that specified exactly how it works so people could learn from and how you're supposed to construct the programs and so forth. There was of course a big effort at the same time to build the .NET framework, all the libraries that come along with the framework, which effectively is the runtime for that language. And that was ongoing in parallel. And that was also a team of... Well, it was bigger than the language team. And all up the .NET organization probably was in the hundreds, because then there was the runtime itself with the bytecode interpreter, and the garbage collector, and all sorts of stuff. And I sort of got involved. I mean, my day-to-day goings was specifying the language and running the design team. But I got involved in a lot of the design of the runtime too, and the frameworks, and the interoperability story, and so forth.

Becky Monk: What were the biggest considerations you were taking into effect as you were creating a new language?

Anders Hejlsberg: Well, creating a new language is an opportunity that rarely comes along. And I often joke, the world needs another language like it needs another hole in the head. I mean, no one needs a new language, because that means everything starts from scratch. Yet new languages are the way whereby we make progress. And so I think it was important to realize that, well, okay, if you're sitting out to create a new language, you got to make sure that you really clean out the broom closet at that point, and get rid of all the skeletons and whatever, and adopt all the learnings that you possibly can from contemporary trends and what's happening in other languages in the industry. And that meant things like object orientation, automatic garbage collection, exception handling, first-class support for things like properties and events that are key in building the modern rapid application development, or websites, or

what have you. Metadata and reflection, the ability for programs to dynamically introspect on themselves, or dynamically generate themselves. Lots of things that you sort of have to factor in.

But then also things like building a language that actually has native support for writing applications in the enterprise space. Oddly, most programming languages, say Java included, or C++ included, actually don't have support for financial applications that need precision beyond 15 digits. Which kind of sounds stupid, but you try to express, say, the wealth of Norway's investment fund in dollars and cents, well, that's more than 15 digits. Norway's a very rich country, with its oil and whatever. But this was relayed to me that this is actually a real problem that surfaced at some point. They blew through what could be represented in a 64-bit floating point number. Plus those 64-bit numbers have round-off errors and whatever. And so we built a language that had built-in support for decimal representation with much higher precision and no round-off errors and so forth. These are just sort of things that were on the plate, right?

Becky Monk: Right. Did they come to you with a number of these Norway?

Anders Hejlsberg: No, that happened way later.

Becky Monk: Were there customer specifications that they had in mind that, okay, if we're doing this new thing, we have to do X, Y and Z to advance it and to make it a futuristic language?

Anders Hejlsberg: Well, I think for sure there was a requirement that it'd be a compiled language. It couldn't be an interpreted language. I think there were, I'd say, pretty strong requirements that automatic garbage collection, object orientation, exception handling were, because that was just where the world was going. The problem with when you write apps in C++ is you have to do manual memory management. And memory management is hard, and people get it wrong. And then you have this null pointer that you dereference and the world blows up.

And so having that built-in gives you an extra level of security and eliminates a whole class of bugs, where you instead can have your programmers spend their time on the algorithms, their creative part of application development that actually brings value, right? There's really no value brought by debugging programs. And so to the extent that that's most of your development effort, it's a waste. And so having something like automatic garbage collection, it's a wonderful way of trading off, say, maybe 10% or 20% of performance for way more programmer productivity. And of course, there was also, I think, an expectation that this product be very interactive in its usage model. We needed a modern IDE, we needed tooling, we needed that quick iteration cycle, the inner loop of the dev cycle where you're writing your code and then you say now you want to run, it just runs right away. You don't have to go through a bunch of hoopla.

Becky Monk: When you were done with this, when .NET was released, what was the industry like? How did it change things?

Anders Hejlsberg: The first preview of .NET was released at the PDC in 2000. I remember, because I was-

Becky Monk: What's PDC?

Anders Hejlsberg: The Professional Developers Conference. I remember being on stage with John Shewchuk and presenting our vision for .NET, and at the time, XML was the buzzword also. Well, first the world, we'd gotten to the internet age where it was humans using browsers to talk to computers somewhere else through the browser.

But the newfangled idea was computer to computer interaction over the internet. The ability to do REST services and remote calls powered by the internet for computers to speak to each other over the same infrastructure that we used for browsing. Basically, over the HTTP infrastructure, and XML was the new thing for that.

.NET was branded as the native for XML-based REST services runtime at that time of release, even though that wasn't necessarily what we had built it for. But sure, it worked well for that. I remember doing a lot of presentations about all of that infrastructure.

This is when ASP.NET became our web server for .NET and where we dramatically changed how easy it is to build websites and to build REST services. That, I think, was a big change. That was really not a thing that existed before. That all of a sudden furthered a

whole new kind of app that you could build, these back-end cloud services. That was the beginning of that.

Becky Monk: How did that change? It definitely changed the computer industry. How did it change the rest of the world? Because now people could develop apps for the internet. How did that change in the bigger scale of things?

Anders Hejlsberg: Well, I think I would like to say that the thing that's always driven us in the developer division is making programmers more productive. Making it easier to build apps that have real world impact. I think for ordinary people, this advent of HTML markup and the ability to put code behind it and build your own websites and literally build your own E-commerce, it just opened up.

It completely changed the world, and it was the early days of the way the world looks now where that's all we do. Now, well, our personal computing device is the phone now in a sense, and we walk around with connectivity at all times. The ability to interact with computers and systems everywhere in real time, do commerce, do social interactions, do email, do video, entertainment. It's all powered by back ends that are built using the kinds of tools that we were launching at that time.

Becky Monk: Yeah. I think just hearing all of this and just putting all of that into context has made me just sit there in awe of how far we came and how far you personally had to go to get there, to get us to where we are today with the phone in our hand.

Anders Hejlsberg: Well, the phone, I'm not going to take credit for that. Honestly, I don't think Microsoft has done particularly well in mobile. It's all up, but that doesn't mean that our infrastructure hasn't been good. The backend infrastructure that we built over time and the adoption that we've gotten by the enterprises of the world has been truly astounding.

To me, that is what brings me to work still to this day every day is the knowledge that millions and millions of programmers out there use your tools every day, and that all of the infrastructure is built using stuff that you created. That is very satisfying, obviously.

Becky Monk: Okay. .NET launches. What was next on your Microsoft horizon?

Anders Hejlsberg: Well, what was next was making it successful, because the first 1.0 of a product can be fantastic, but it is also full of things that you didn't get to. We had a long list of things that we knew we wanted to do to mature the language. I've always been a believer in playing the long game, in sticking with a project, not just for one release and then moving on to something else, because these development tools are different.

I stuck with Pascal and Turbo Pascal for a decade, and Delphi for an overlapping decade. C# I was with for over a decade, and you kind of have to in order to truly make great products. It's just not possible to have a perfect programming language and development environment surface out of thin air in one release.

We set about growing a methodology of how to do multiple versions of the product, how to listen to our customers, how to build the right features at the right time. Things like in version 2.0, we did generics, which I'm not going to get into why, but it's a very powerful feature that helps with a lot of programming angles.

In 3.0, we did this thing called LINQ, language-integrated query, where we focused very deeply on the interaction between programming languages and databases and trying to reduce the friction as much as possible, because the kinds of apps that the enterprise was building all ultimately talked to data in some form. They all had a database in there somewhere.

There was this big mismatch between databases and programming languages that had gone on forever, and we had languages like Java and C# or BASIC or C++. Then the databases of the time were all powered by SQL, the query language. SQL has the relational algebra, all sorts of other concepts that have nothing in common with programming languages, and programming languages have object orientation and all sorts of stuff that doesn't exist in SQL.

It was friction, friction, friction whenever you were trying to talk to a database, and crappy tools and overhead. We worked really hard on reducing that, and that got us even deeper into the enterprise and brought us a lot of love from enterprise programmers who are building these back ends that we just talked about.

For every version, there was a theme, if you will, and a process that we went through. We maintained our design team for all the years I was there, had regularly recurring meetings. We have a file that contains all of the design notes going back to 1999 of everything, which was wonderful because inevitably people would arrive and they'd go, "Oh, we need this."

Then you go, "Gosh. God, we talked about that a few years ago?" Then you go search for it and there it is. There was an institutional memory of our reasoning for why the product looks the way it looks, and all of that pays back over time. It doesn't pay back in one release or another, but in multiple releases, that's really the only way to play that game in programming languages.

Becky Monk: Well, I want to be mindful of time, because we could be here all day, but you keep talking about Bob Muglia. He's coming in next.

Anders Hejlsberg: Oh, wonderful. I'll have to say hi to him. I haven't seen Bob in a few weeks or months probably.

Becky Monk: I want to skip ahead. What was the next big project that you did work on? After the decade or so, what was next for you?

Anders Hejlsberg: Well, it's funny because if you'd asked me at the time, I didn't know that there was anything next, but I was starting to get interested in maybe looking for another opportunity. But after you've done something for 10 years, you get a certain itch, an urge to maybe try something different.

I think the genesis of it was at one point what was happening at the time, we're now talking about 2010. What was happening at the time was increasingly, we had the big battle with Netscape over the browser for 10 years in a sense, and we'd won the battle. But then in a sense, we'd also deserted the playing field.

Do you know what I mean? Nothing much happened in browsing and the JavaScript programming language that was built into browsers, but then eventually it started dawning on everyone that really, the true cross-platform language wasn't actually Java, because Java doesn't run on all these clients.

The thing that runs on all the clients is JavaScript, and really the true cross-platform language is JavaScript. JavaScript was all of a sudden becoming very important. HTML5 had happened, and all a sudden the presentation surface in the browser had become much more capable, and better runtime implementations of JavaScript was being built, like Google's V8, which had a JIT compiler and all of these newfangled things that all of a sudden made your JavaScript run 10 to 100 times faster.

All of a sudden there's this platform inside the browser that allows you to build real apps with a completely friction-free deployment model that runs everywhere, including on mobile devices and iPads and whatever. Of course, everyone was getting wind of that, and they were going, "We need to write apps that run in the browser. The browser is the new application platform."

Except JavaScript is a horrible programming language. It's a dynamic language. It harkens back to, in a sense, the same principles as these early basic interpreted languages. It's very hard to build good tools for JavaScript, because JavaScript lacks some of the fundamentals of more modern programming languages, like Java or C#.

It lacks a static type system, and static type systems are what allow programmers to express their intent with the program. They don't just create a variable called X, and then you can put anything you want into X. You can put strings or numbers or lists or objects.

No. In grown-up programming languages, you not only say, "I want an X, and I want X to be a number. I want you to assume, and I want you to tell me if I ever make a mistake and not putting numbers into X or whatever, and I want to be able to describe to you what my data structures are supposed to look like, so you can validate that they actually look like that and catch the errors before I run the program."

It's better to find the errors before the space shuttle flies than whilst it's flying. Compiled languages were super important in the enterprise. Yet on the front end, all you had was JavaScript, which is interpreted and does not have a static type system.

People were doing the oddest things in order to write their apps. For example, I think the genesis for the TypeScript project, which is where I ended up getting involved, was the folks building

Outlook.com coming to us and asking whether we could please productize this technology called ScriptSharp.

I'd heard about it before and I'm like, "Well, what is ScriptSharp?" Well, it's this thing that allows us to write in C#, and then we run this cross compiler that turns our C# into JavaScript, and then we run that in the browser. I go, "Now, why would you want to do that? That sounds like a completely backwards way of writing your JavaScript apps."

"Well, it's so we can get grown-up tooling. It's so that we can have a great integrated development environment. We can write data types and definitions for our programs. We can have source code control. We can have all of these things that go with big projects. Basically, it's so that we can have a whole team work on the app."

That was really interesting. I hadn't paid that much attention. But then you go, "Wow. Is JavaScript really that bad, and is that really the way to fix JavaScript is to actually use another language and then just use JavaScript as something you never see?"

We set out to understand what is it that's wrong with JavaScript and how can we make it better, and how can we build this development experience that these people are expressing to us that they desire, but that they don't have, and therefore they're doing this other thing. That was what became TypeScript, which was really, "Let's fix JavaScript, if you will, and build a wonderful development experience for the JavaScript developer."

Becky Monk: How many years were you on that project?

Anders Hejlsberg: Oh, I'm still on it. I'm still on it. It's still ongoing. Yes. Our first official release was in October of 2012. The core early founding team, myself, Steve Lucco, and Luke Hoban. We started, I think, in December of 2010 to come up with the initial ideas for what is this going to look like, and what is it that we're going to try and build?

Of course, that also set us on this whole new journey called open source. We full well knew. You've got to remember, at that time in 2010, Microsoft was basically the evil empire, particularly in the eyes of the open source developer community. JavaScript was very closely aligned with open source, and there was no one in that community who would give us the time of day. We were just like, "No, I'm not ever, ever dealing with Microsoft."

We faced a massive uphill battle here. Still, we knew we had to go there though, because that's where the world was going. We had to learn how to do open source, and that was a big cultural learning process, a very big culture, and a very big shift for Microsoft.

It was very interesting to be a part of that, because the first thing we realized with this TypeScript thing was, there is no way we're ever going to appeal to anyone in this community with a proprietary product. None. It has to be open source, and it has to be open source from the get-go.

Just getting the corporate buy-in for that was, oh, my God. Even though we were talking the talk, it was real hard for us to walk the walk. There was just a lot of institutional just, "I don't know about that. I don't know that we're going to give away our IP. Oh, boy. That doesn't sound right."

It's like, "What do you mean? Anyone can see our source code?" It was so ingrained in us that source code IP, that was something you kept in the safe and only revealed to the inner circles, because that was the crown jewels, and this was all about giving away the crown jewels on day one.

Becky Monk: How did you guys convince finally that this was the way it had to be?

Anders Hejlsberg: Well, it wasn't easy, but I think the argument that effectively works is that we all know that JavaScript is where it's going to go, and browserbased apps is where it's going to go. We also know that if we don't go there, someone else will go there and then they will own it, and we will just forever be out of it.

We have to get there and we have to play by those rules, and that means we have to get to understand open source, because that is also where developers were increasingly voting with their feet. The trick was how do we find a way to go there without giving away our entire business?

Now, it turns out that there are parts of your business that you can build open source and gain mind share, and then there are services you can sell on top of that that are valuable to people, such as running their code in our data centers, in our cloud, and whatever that takes, big capital investments to build and whatever. No one expects to get that for free.

But developer tooling was effectively going pro bono. I said at the time, and I will say it to this day, I don't believe there will ever be a successful programming language created. That was true 10 years ago, but now there will never be another commercial programming language.

They're all going to be open source, because if they're not, programmers just don't care, because they're just, "Well, then I'll just look at one of these other open source languages." It's a done deal that has already played itself out, and so we had to go play in that sandbox and it was fun learning to do.

It's very liberating. It's actually a wonderful way to build software, I got to tell you, because the one thing that it does is it puts you right there with your users. Once you move your development processes onto GitHub and everything, you're talking to your users every day.

Your users can see what you do in the product. They can see how it's put together. They can suggest how you might fix this thing, or they can actually build the fix and ask you, "Would you please merge this?" It's completely friction-free compared to our

traditional process, which was we'd ship maybe once a year, and the minute we shipped, then we would open up and then everyone would check in their code and completely break the product.

Then we'd spend a year trying to put it all back together, and then we were trying to convince people to beta test this, but they had to sign all these papers to get a beta. "Well, we can't sign all these NDAs." No one tested it. You'd ship it a year later, and only then would you find out all the stuff that's broken.

It was just a dumb process that made everything go slower and make the products be worse. A lot has changed there. It's so joyful to work in that space where code is king. Do you know what I mean? Everyone can participate and build the best possible thing for everybody.

Becky Monk: I love that. I love that. You're still there. You're still innovating. Tell me what it is that you are most proud of your time at Microsoft so far.

Anders Hejlsberg: Well, I think it's the impact. It's knowing that I had a big part of creating two of the world's five most popular programming languages, and they're in use very actively today. That's very satisfying, and it just makes you feel like the work you do is relevant.

We all want to be relevant. We all want to do stuff that the world cares about. That is ultimately the thing that gets you going. At

least to me it is. That's probably it. Knowing that you've made a difference in your work.

Becky Monk: Well, you definitely did. You definitely are. Talk to me a little about the culture at Microsoft, because you've stayed there for more than 40 years.

Anders Hejlsberg: Well, I've been in the industry for 40 years, but I've been at Microsoft for 28 years.

Becky Monk: That's a good chunk of time to be at one company.

Anders Hejlsberg: It is, it is.

Becky Monk: What is it about the culture and the company that make you want to stay where you are?

Anders Hejlsberg: I'm an engineer, and Microsoft was always driven by technology. Microsoft is not a sales-driven company. To me, that is probably the thing that has motivated me the most to stay. It's thinking, "Would I go to these other?" You look at companies like Google or Meta and whatever, and really, we all know who's king. It's the buyer of the advertising.

We know who the product is. The users are the product, and eyeballs is what we sell. It's not really technology. Technology is the enabler for that, and of course, it's important. But I love the fact that we build products that people pay money for, because it's technology that helps them do their job better. It's making the enterprise run better. It's making cloud run better. It's making AI run better. All of that is powered by programming. I think to me that feels really good, I think. I like that. I like that a lot, yeah.

Becky Monk: When you think about the innovation that's going on today, I know the technology happened a while ago, but we're all catching up on the outside to what's going on today. How does it feel to be part of that ongoing innovation?

Anders Hejlsberg: Exciting and exhausting at the same time, right? I mean, the whole AI phenomenon, no one saw that coming really. I mean, you go back five years and it's ... well, there maybe were inklings of it in research papers or whatever, but I think it took everyone by surprise, what that stuff can do and how it's, yet again, changing the world of software and the world of technology. It's yet another fantastic new tool that we're going to put in our toolbox, yet it's also a beast that we have to tame and we gotta worry about all of the bad ways in which it can be harnessed, just like all the bad ways the internet could be used, and then satellite technology and what have you, right?

It's exhausting and exciting all at the same time, but that is what this industry is. I mean, you can never stop in this industry. The minute you feel like, "Oh, this has gotten real nice and easy and I don't really have to," then that's because you've become irrelevant,

right? I mean, that's really what is the case in technology. Just you gotta ... you gotta keep running.

Becky Monk: That's great. Can I talk a little bit about Microsoft's philanthropy, the giving campaign and that? Is that something that you participate in or that has a place in your heart?

Anders Hejlsberg: Yeah. I've never really used the official giving tools in the giving campaign, but the matching program has been wonderful. Our family tries to use that as much as we can in our philanthropic giving. That has been awesome, to have that ability to double your impact. It makes a difference. It really does. I mean, the joy of giving money away is fantastic. I mean, it really is. We've given a lot to the arts and to educational institutions and whatever, but I wouldn't say I'm ... I'm still a working guy, do you know what I mean? That's what keeps me going, but philanthropy, I think it's wonderful that we do that, yeah.

Becky Monk: Are there any fun stories from the early days of being in the room with the other developers, and a challenge that you guys came up against that you just had to work crazy hours? Or what is it that you look back and think, "Oh, that was a great time, that was a great moment"?

Anders Hejlsberg: I think there have been many of those. Being on stage at the first PDC in 2000 and announcing this entirely new platform to the world was, I mean, what a moment to be part of that. I suppose another one, I remember when we open-sourced all of C#, through

the Roslyn project, where I literally on stage pushed the button that made the GitHub repo public. That was a fun moment.

Other moments, I mean, being in Bill reviews, like reviews of your project, and every Bill review always came with, you know, Bill would always sit in the same place at the table. He always immediately would flip through all the slides and then write a bunch of notes and you have no idea what he was writing down, but he wouldn't say anything. Then all of a sudden he'd call you on ... often he came out like, "That's the stupidest thing I've ever heard," right? Then that was your call to attention, and now you need to tell Bill why it is not the stupidest thing he's ever heard, and then why in fact he is wrong about his assumption about this or that or whatever. That challenge was interesting, right, and it was fun to be in the room for those.

I've seen people handle it both well and not well, but that was the culture of the early Microsoft, right? I mean, that was in many ways your ... it was both a good thing and a bad thing, because Bill is ultimately, at the time, was a technologist and the company was run by technology, but it could also get to a point where it was almost like this navel-gazing introspection, and really all you worked for was the Bill review as opposed to the customer. Do you know what I mean? We've had projects where we've gone a little bit astray on, "Well, where exactly is the customer value in this unification of this and that technology," or whatever, right?

Becky Monk: Probably countless stories. Well, I know we're getting short on time, but just before we stop, I want to ask is there anything else. We were doing 28 years in an hour and a half. Is there anything else

that you want to chat about that you really think we should dive into a little bit?

Anders Hejlsberg: That's a good question. I don't think so, honestly. I haven't really thought of it, yeah.

Becky Monk: I know we talked about what you're proud of, but is there a legacy that, when you do leave the company, that you hope people will think about with you or remember about you?

Anders Hejlsberg: Well, I mean, I hope. I've worked with so many great people on this, and for me, and perhaps the one thing I should talk about, is for me it's sort of been this journey of learning to become a team player. That is probably the thing that I feel like I have learned, 'cuz I started out as a singular perfectionist who wanted to do everything himself, right, and it doesn't scale and you can't do that. You have to constantly fight your, "Just give it to me and I'll fix it," you know, because you can't do that, right? Learning that is something that I still do every day. I still learn how to be a better team player, and I enjoy that.

Becky Monk: Fantastic. I'm going to holler around to everybody and say, okay, what else do we need to do? What else should we be thinking about?

Anders Hejlsberg: What was it about the first computer that really sparked my imagination?

I decided to work in computers because at heart I'm an engineer, and engineering was always something that was in the blood of my family as well. My dad was an engineer, and so I knew pretty early on that that's where I wanted to go.

Becky Monk: How did Turbo Pascal then become that revolutionary product?

Anders Hejlsberg: I think Turbo Pascal changed the industry by being the first commercially available integrated development environment. If you look at development tools today, that is the prototype for what they all look like.

Becky Monk: What was the next project you worked on at Microsoft?

Anders Hejlsberg: Well, the next project then was C#, which was about building a language that had the power and expressiveness of C++, but with the ease of use and rapid application development capabilities that Visual Basic had at the time. Then add to that modern concepts like garbage collection and exception handling and object orientation properties and events and so forth. That was my next project.

Becky Monk: What were you most proud of, of your time so far at Microsoft?

Anders Hejlsberg: Well, what I'm most proud of with my time at Microsoft is that I got to participate in the creation of two of the world's five most popular programming languages today, and that I have millions of programmers use these tools every day to build software that powers the world, and that is just immensely satisfying.

Becky Monk: What is it about the culture of Microsoft really that made you want to stay and excited to stay for 20-plus years?

Anders Hejlsberg: The thing about the culture of Microsoft that makes me excited and always excited me is that it's a company driven by technology, not a company driven by sales.

Becky Monk: For somebody who doesn't know, what is the significance of a compiler?

Anders Hejlsberg: The significance of a compiler is that it is this tool that can validate your program before you run it, and find errors or bugs before you actually run the code. It can also translate your code into more efficient machine code that could be executed by the CPU and run much faster than, say, an interpreter would otherwise.

Becky Monk: Did you have any idea how much Turbo Pascal would grow?

Anders Hejlsberg: Did I have any idea how much Turbo Pascal would grow? I had absolutely no idea. I mean, it became huge in retrospect, but we just thought it was a good idea. We thought this was the way development tools should look. Of course, we thought it worked really well, but we couldn't have imagined in our wildest fantasies how big it got, no.

Anders Hejlsberg: The first company that I got involved with in Denmark was called PolyData, and it was the first company in Denmark to actually open a computer store where you could walk in and buy a kit computer, just as a hobbyist.

Becky Monk: How did you end up working with Borland?

Anders Hejlsberg: Well, so at PolyData we had built what is basically the predecessor of Turbo Pascal. We had a product called PolyPascal, it was called, which was our basically version of Turbo Pascal. It had a compiler, a

runtime library, an onscreen editor. One of my coworkers knew this other Danish company that wanted to get into software. They had founded this company called Borland. They had a couple of products written already in Pascal, but they were using this other Pascal compiler called Pascal/MT+, which honestly was pretty horrible compared to our product.

My buddy explained to them how, "Oh, we have actually this Pascal development environment that is way better than that, you really should give it a shot and check it out," and gave them a copy of it. Within a week they came back and they were just floored, and suggested that we do a deal where we jointly take our product and change it around and make it into what became Turbo Pascal by adding a new WordStar-compatible editor and changing the runtime library, writing new documentation for it, but that's basically how we started.

This company had ... I worked with the three founders, Niels, Mogens and Ole, all Danish, and they had a few other programmers and they were in Copenhagen, and we worked together. That was in '83, you know, for about six months, to repurpose our product and then launch it as Turbo Pascal in the U.S.