**ANDERS HEJLSBERG:** (VOICEOVER) Surely. Anything can be done here. Look at this diagram. I just drew it for you. Just go and make it so, right?

No, coding is hard and it continues to be hard, and code gets bigger and bigger and bigger but our brains are not getting any bigger, and this is largely a brain exercise.

**KEVIN SCOTT:**  Hi everyone. Welcome to Behind the Tech. I'm your host, Kevin Scott, Chief Technology Officer for Microsoft. In this podcast, we're going to get Behind the Tech.

We'll talk with some of the people who made our modern tech world possible and understand what motivated them to create what they did. So join me to maybe learn a little bit about the history of computing and get a few behind the scenes insights into what's happening today. Stick around.

**[MUSIC]**

**KEVIN SCOTT:** In this episode of Behind the Tech, we'll meet Anders Hejlsberg. Anders has always been one of my coding heroes. He built Turbo Pascal, at Borland, which is the tool that I use to become a real software engineer.

At Borland, he had a long career where he was Chief Architect of Delphi in addition to the Turbo Pascal tools and eventually moved over to Microsoft, where he helped create C# as the Lead Language Designer, and today, he spends his time as the Core Developer on TypeScript.

Anders has had a 35-year career building development tools that software developers love. We're going to learn how he's done that today on Behind the Tech.

**[MUSIC]**

**KEVIN SCOTT:** So, welcome, and thanks for being my first guest on this debut podcast.

**ANDERS HEJLSBERG:** Thanks for being willing to experiment.

**KEVIN SCOTT:**  You were a calculated first choice because, I think I've told you this before, but I learned to program when I was 12 years old. My entry was BASIC in 02 Assemby language, the 80 Assembly language. The first real programming I ever did was on Turbo Pascal. I went to a science and technology high school and took Intro to CS class and Turbo Pascal 5.5 was the tool. I don't think I would have chosen a career in computer science if it hadn't been for that sort of confluence of things. So like in a way, you're responsible for my career.

**ANDERS HEJLSBERG:** That's awesome.

**KEVIN SCOTT:** The main reason I wanted to have you on the show is this must be the case for a huge number of folks and we'll get into your journey as an engineer. But what made you decide that you wanted to build programming and development tools? Because that's sort of been your entire career.

**ANDERS HEJLSBERG:** Yes, it has. You know I've been doing it for more than 35 years now, and it's kind of scary to think back and it's such a long time ago now. I mean, and the world and the industry, there wasn't really an industry even. IT was so different. I mean, in the industry, I got started probably in the early '80s, but I got started coding in high school back in the late '70s. Probably I was trying to think back '78 or, yeah, '78.

**KEVIN SCOTT:**  Were you the type of tinkerer where you were building a little programming language?

**ANDERS HEJLSBERG:**  No, no. So I was born and grew up in Denmark, and I went to a high school outside of Copenhagen. It was one of the first high schools to offer students access to a computer.

**KEVIN SCOTT:** Yeah. What was the computer?

**ANDERS HEJLSBERG:**  There was the old HP 2100 it was called. It had 32K of ferrite core memory. You can literally open it and see the ferrite core. So it was amazing. Paper tape reader and then after a while, they got a 14-inch one megabyte hard drive, which was absolutely state of the art.

**KEVIN SCOTT:**  It was like huge.

**ANDERS HEJLSBERG:**  It was enormous. At the read-write head on that thing had this gigantic magnet, and everything was so primitive. You could program it, and we put that poor computer through so much torture, like trying to make music by moving the read-write head on the hard drive. They would vibrate the whole tape, and it was-.

**KEVIN SCOTT:**  Yeah. Yeah.

**ANDERS HEJLSBERG:**  So that was sort of a very hands-on introduction to computing, and I too remember that you could either program it in Assembly code. I think there was a FORTRAN compiler but then there was an HP ALGOL compiler.

**KEVIN SCOTT:** Oh, Interesting.

**ANDERS HEJLSBERG:**  That was actually sort of how I vectored into that branch of programming languages ultimately. Because that was the first language I was taught. We didn't have BASIC. There was no BASIC on that computer. Do you know what I mean? So I learned ALGOL and this was a very primitive compiler. I mean, it didn't support recursion.

**KEVIN SCOTT:** Right.

**ANDERS HEJLSBERG:**  That machine had no stacks so call instructions would just store the return address and the first word and then return was an indirect jump back to that. Right? I'm probably going to get in trouble for saying it, but better than FORTRAN or COBOL.

**KEVIN SCOTT:** They were still better than FORTRAN.

**ANDERS HEJLSBERG:**  Yeah. Yeah. But anyway-.

**KEVIN SCOTT:** How much were you all influenced by the Scandinavian programming languages? So Niklaus Wirth was the designer of the Pascal programming language and then MODULO later on.

**ANDERS HEJLSBERG:**  It's funny, in high school, I didn't know anything about any of that, and really, it was just sort of learning by doing. Then, I started electrical engineering at the Danish engineering academy. There, I met a guy who had just transferred from another school into the engineering academy and we both liked playing cards, and he lost some money. So I had some IOUs so I had to get to know him. So I ended up getting involved with him and starting a computer company in Copenhagen. We had the very first retail store where you could go in and buy a kit computer.

**KEVIN SCOTT:** Wow.

**ANDERS HEJLSBERG:**  We were right at the cusp at that time of microprocessors becoming democratized. The 8080 and the Z-80. I'm like this is when the Sinclair ZX80 started to happen. There was this British kid computer called the NASSCOMM, which had a Z-80 in it, but I started writing software for that one then. It came actually with a Microsoft ROM BASIC.

**KEVIN SCOTT:** So what sort of software were you writing?

**ANDERS HEJLSBERG:**  Well, so first, I started writing games. It's funny, I never liked playing games but I liked writing games.

**KEVIN SCOTT:** Were these sort of text-based adventure games?

**ANDERS HEJLSBERG:**  Yeah. It was like Moon Lander and Star Trek and that sort of thing, right? You could actually buy these books. I think it was like Byte Magazine had these 101 computer games where you could type it all in, right?

**KEVIN SCOTT:** Just great. Like it was such a great thing.

**ANDERS HEJLSBERG:**  It was awesome. So I learned a lot. But then, I started actually getting into assembler coding and sort of trying to understand more about how the machine worked. Then I got curious about extending the Microsoft ROM BASIC. Because that ROM BASIC, it had to fit into an 8K ROM. So there were a bunch of commands it didn't have. There was no renumber command, which was a royal pain in the neck, because if you ran out num now, you had to manually go retype every line. But there were some extension points where you could actually sort of hook into the ROM BASIC and because they were an extra slot for E prompts on the motherboard. I wrote this little 4K ROM BASIC extension that gave you renumbered and a bunch of other things. So that was like a little plug-in tool kit. Right?

**KEVIN SCOTT:** How old were you then? This is like after university?

**ANDERS HEJLSBERG:**  Yeah, I was probably 20.

**KEVIN SCOTT:** So like very, very early in your career you were mucking around with your programming tools.

**ANDERS HEJLSBERG:**  Sure. Yeah.

**KEVIN SCOTT:** Is that just sort of a sensibility thing or-

**ANDERS HEJLSBERG:**  I just sort of found it interesting. There are a lot of challenges there, right? Then that interest further got piqued by- now that I've figured out, “Oh, you could actually the extend.” I remember still missing that ALGOL programming language and I was telling my buddy, “Maybe I should write a little ALGOL thing that we could probably-.” “No, no. There's this new thing called Pascal. You really should check that out.” “Of course. Okay. Okay.” Then I started like, “Oh, this is better.” Because one of the things that Niklaus did consistently through his entire career, he's the creator of Pascal, but he was involved with ALGOL and then later MODULO and Oberon, and every language got progressively simpler.

**KEVIN SCOTT:** Yes.

**ANDERS HEJLSBERG:**  ALGOL had all these complex call by ­value and call by name and then Pascal dumped a bunch of that, and then MODULO dumped even more of that and that Oberon got even more minimalistic.

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:**  I remember I've never written a real ALGOL program but because I was a compiler guy, the ALGOL grammar was all over the place.

**KEVIN SCOTT:** It was complicated.

**ANDERS HEJLSBERG:**  So complicated. So anyway, so there, I set to work and I wrote this little Pascal that was effectively the nascent Turbo Pascal, if you will, right? It had a little onscreen editor and a runtime library and a compiler and it was all squeezed into 12K in a ROM.

**KEVIN SCOTT:** What gave you the idea to do that? Because that's sort of a new thing, right? That might have been the first ID, right?

**ANDERS HEJLSBERG:**  Yeah. That was probably one of the very first IDs ever. Yeah.

**KEVIN SCOTT:** That's amazing.

**ANDERS HEJLSBERG:**  Well, UCSD Pascal was on the Apple too, but it was sort of different. It was a PCODE interpreter and the ID was not really quite as user-friendly. It was more like a sort of semi-command liney ID thing.

**KEVIN SCOTT:** So you wrote this thing in mostly in Assembly?

**ANDERS HEJLSBERG:**  It was all in Z-80 Assembly. Yeah. Oh, yeah. Now, that was my weapon of choice at the time, and that grew into a CP/M-80. It was obvious that one was a subset implementation, and then that grew into a fuller subset and then finally a full implementation for CP/M-80. At that point, we met the guys. Borland was founded in Denmark. A lot of people don't know that. We knew the original founders because they had had another computer company and it was a very small industry in Denmark. They were writing some stuff in Pascal MT Plus. I remember from digital research that CP/M.

That was a horrible Pascal. It was like one of those insert-pass-one disk in drive. Right? Then they would grind and then you'd take that out and then it would write some of the temporary code and then you'd insert the code generator disk. It took an afternoon to compile a small program.

**KEVIN SCOTT:** There were like dialects of Pascal that were just like not as nice as they were Turbo Pascal.

**ANDERS HEJLSBERG:**  Right. Then they were afraid of making extensions, and then, I was never really afraid of making extensions if they were useful.

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:**  So these guys were “Wait. We got actually this implementation you should check it out.” They didn't believe a word of what we are saying because that's not possible.

**KEVIN SCOTT:** I just want to double-click on this point again. Coming up with one of the first integrated development environments that you have written in Z-80 Assembly language at that point, that's a unbelievable breakthrough.

**ANDERS HEJLSBERG:**  I suppose in retrospect, yes. I've never really thought of it that way. But, you know, it's-

**KEVIN SCOTT:** Just incredible. [inaudible].

**ANDERS HEJLSBERG:**  It just seemed like this is going to be so much better

than having to have a first date a load an editor and then load at the bargain. Why not just put it all to there? I don't know. I never really-

**KEVIN SCOTT:** And especially, at the time, because again, like more framing. These are not Windowed systems, can't have multiple things opened at the same time. It's super tedious to switch from one program to another. So like having everything in one place is just as huge.

**ANDERS HEJLSBERG:**  Totally. The added compile, run, debug, cycle, just shrunk by many orders of magnitude.

**KEVIN SCOTT:**  Yeah, and I'm embarrassed to say I've forgotten what was it F9 to compile and run, or was it F5?

**ANDERS HEJLSBERG:**  I don't even remember what it was. I think it was F5 yeah but.

**KEVIN SCOTT:**  It was like miraculous.

**ANDERS HEJLSBERG:**  Maybe F3 but yeah, it was great. There were all sorts of tricks in there like the runtime library was the first 12K of the system and then when producing code, I just copy the first 12K into the X we were producing. There is your runtime library, right? Then generate code from there on out and you could compile the memory and we'd put the code in memory and run it, right? Or the original implementation compile to tape, to floppy tape and then you go, sorry to the tape recorder interface, right? Then you can load that machine code up because I mean there was only K of memory. I mean it was crazy.

**KEVIN SCOTT:**  Yeah. So, I bought a copy of Turbo Pascal 5.5 out of a catalog called Programmers Paradise. This is just sort of how you used to buy software and so I forked over my two hundred bucks or whatever.

**ANDERS HEJLSBERG:**  Oh no, it wasn't even that. It was $49 like $49.95.

**KEVIN SCOTT:** It was affordable because I was poor. So thank you for making cheap software.

**ANDERS HEJLSBERG:**  Yeah.

**KEVIN SCOTT:** But like it came on like this bundle of five and a quarter inch floppy disk that sort of fed into my, and I was lucky...

**ANDERS HEJLSBERG:**  I mean I got to get credit there to the Borland guys, to Felipe and Neil and Mogens and all of the original founders of Borland because what they did, they ended up licensing this Pascal compiler that I had in my small company and we were selling it for 500 bucks and doing business the way business was normally done for a software, right? Then they licensed it.

Of course we never thought to put a minimum price on what they were supposed to sell the product for and then come back with we're going to sell it for $49.95. What? You got to. It's just terrible. It's supposed to sell for 10 times. And so they cut the price by 10x and then they literally sold four or five orders of magnitude more copies of it, right? That was tremendous.

**KEVIN SCOTT:**  Yeah, I mean when you think about the life cycle of innovation, it's like not always just about the technical thing. Like that wouldn't have been an appropriate environment for students if they cost 500 bucks.

**ANDERS HEJLSBERG:**  At that time it was like, hey, if you only charge 50 bucks, why even pirate? It was worth just to have the manual that actually tells you how it works, right?

**KEVIN SCOTT:** Yeah. So one of the other things that was really good about Turbo Pascal was the manuals were actually quite good. How big a push did you have in that?

**ANDERS HEJLSBERG:**  I had written manuals for the Pascal compiler that we had in our company called Poly Pascal and it had a pretty extensive manual for the language. The introductory parts were not as great and Oler, who wrote the documentation, Oler Rassmussen. He did a great job on turning that into something that was really a fun book to read and so a lot of people were very happy with the documentation as well.

So it was just a confluence of a bunch of factors that just came together right. And the right thing at the right time and the industry was so nascent. I mean that was it.

**KEVIN SCOTT:** So going back to your university days, did you have a professor or colleagues or like someone who was really great at the programming language the compilers part of the curriculum where they sort of lit this aha moment for you.

**ANDERS HEJLSBERG:**  Not a whole lot actually. I did electrical engineering because in order to do computer science, you would have gone to the university. But I was always more of an engineering sort of focused and I wasn't necessarily sure that programming was what I wanted to do. Do you know what I mean?

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:**  At the school at the time, they had yet to introduce personal computers. So, they had the Northern European Computing Center there where you would turn in your stack of punch cards and then the next day you'd get the printout which would just be 100 pages that said error, error, error and then you'd have to go back figure out why it was that. It was horrible it was like a week long debug cycle.

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:**  But then they started getting some again also some Z-80 and CPM-based microcomputers and by the time I left the university there, they were actually using Turbo Pascal or Poly Pascal the one that I written in the introductory courses.

**KEVIN SCOTT:** Holy crap. That must be great.

**ANDERS HEJLSBERG:** So my professor did allow me to not attend the class he gave me an A.

**KEVIN SCOTT:** Because you've written the compiler.

**ANDERS HEJLSBERG:** Yeah, that's right.

**KEVIN SCOTT:** That's great.

**ANDERS HEJLSBERG:** Yeah.

**KEVIN SCOTT:** I think one of the things that folks may have started to take for granted now is like you sort of jump online, you've got Google, you've got the open source community.

**ANDERS HEJLSBERG:** Right.

**KEVIN SCOTT:** YouTube and like all these video resources. I would argue that it's like much easier now to bootstrap yourself as a programmer or even in sort of esoteric things like how to build a compiler. So, how did you get your bootstrap, like how did you figure out how to do this stuff?

**ANDERS HEJLSBERG:** You learn by doing. You learn the hard way in a sense. I mean in retrospect there were so many things that had I known that, I could have gotten there a lot quicker. A couple of things I suppose is one is, you couldn't stop me from doing it. I wanted to do it.

Someone once talked about how do you get really good at something. Well, it's by putting in the time. There's this talk about the 10,000 hours that you got to put in, and I was like yeah I definitely put in 10,000 hours in the first three years that I did this because I was obsessed. I just have to do it I didn't care if anyone paid me. It was just a challenge of it was so interesting.

**KEVIN SCOTT:** It probably wasn't 10,000 glorious hours. You were probably were making lots of mistakes so.

**ANDERS HEJLSBERG:** Absolutely. Absolutely. Yeah. I didn't know how to write a compiler. But then you started looking at compilers that were written and since my native language was machine code I mean heck yeah I could disassemble other stuff and figure out how that worked right and then you gradually learned.

But there were still things I didn't know. Even in the first version of Turbo Pascal I remember all the symbol tables were just stored as linear lists, which of course doesn't scale so well, right? I mean and then I remember learning about hash tables so I was like, "Holy cow! I got to go home and try this."

Then I put it in and doubled the speed of the compiler, right? I remember that's one of the first books that I thought was really instrumental for me in understanding a lot of sort of the basic computer science was written by Niklaus Wirth called Algorithms Plus Data Structures Equals Programs.

**KEVIN SCOTT:** Yeah. Which is a great book.

**ANDERS HEJLSBERG:** Oh my God. It is the most pragmatic computer science book I have ever seen. I was like never into the symbolism and the formal proofs and whatever. I'm just like give me the algorithm. Tell me. Let me understand how it works, right?

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:** This thing explained how binary trees and binary search and hash table and how to build a compiler with a scanner and a lexer and a parser. Do you know what I mean? Code generator and recursion and oh my God it was like I loved that book.

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:** You can go back to it today and it is still super valuable.

**KEVIN SCOTT:** I've always loved books like that. Another good one Dave Hanson who used to work at Microsoft Research here wrote this book called C Interfaces and Implementations. And I like C because it's sort of simple. I mean it's got a bunch of hair on it, right?

**ANDERS HEJLSBERG:** It does. It does. Easy-to-make mistakes. Yeah yeah.

**KEVIN SCOTT:** It's easy-to-make mistakes not the most elegant language but it's simple you can sort of pack all the rules and you add. But this book, C Interfaces Implementations was genius because it was a bunch of data structures and sort of approaches to programming that were super pragmatic. They had these very broad applications and it was almost like teaching C to do crazy object-oriented sort of things that couldn't-

**ANDERS HEJLSBERG:** All your tables do.

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:** Yeah, yeah no I mean heck at Borland, we definitely had lots of systems that were sort of calm that Windows is largely based on is effectively roll your own read tables, right?

**KEVIN SCOTT:** At Borland, how did you all get feedback? How did you know you were pushing your product in the right direction?

**ANDERS HEJLSBERG:** That's a good question.

**KEVIN SCOTT:** The cycles were long, right?

**ANDERS HEJLSBERG:** They were. Back then there were a lot of trade magazines, and you got a lot of feedback through that. Through the reviewers, like [inaudible] and Byte magazine and what have you, right? We also had bulletin boards early on. Like the old BBSes that we would monitor in our tech support, and so, there were a variety of ways. People would write us letters and send us letters. I have so many letters with suggestions for stuff. So, we always had a long list of things we wanted to do.

It was really not a matter of figuring out what to do, it was like a matter of which ones don't we do, and sorting it and doing the right ones. You mentioned too Pascal 5.5. That was the release where we introduced Object-Oriented Programming. Oh my God, that was like the big buzz word of the time. That was like the AI of the 80's, right?

**KEVIN SCOTT:** It was a good implementation. So, you must've taken some inspiration from Modular.

**ANDERS HEJLSBERG:** Well, Object Pascal was probably the work that Larry Tesler did at Apple. That was definitely an inspiration. Although, our first implementation interwove Pascal 5.5 was a little bit different. It had some influences from C++ also.

Then by the time we got to Delphi, I think we were more true to Object Pascal. There was no difference between object references. You didn't have to use an up arrow to dereference them, and then that made the syntax a lot more gentle on user. It was just fu.bar not fuuparrow.bar.

**KEVIN SCOTT:** Yeah, I remember. So, 5.5 was where I learned Object-Oriented Programming. I think I'm sort of lucky because if you got thrown into a modern Java for instance, and that was your playground for Object-Oriented. It's so complicated now.

**ANDERS HEJLSBERG:** It is, but it's also in some ways simpler, right? Because back then, the languages that people used, by and large, did not have garbage collection built-in, and were very not safe and had minimal runtime checking.

In fact, we ship Turbo Pascal by default with runtime checking turned off, because we did better on the benchmarks that way. A lot of reviewers were not smart enough to know to turn it off, and so, they were comparing apples to oranges. We just wanted to make sure that we were always in the running there. Once you introduce garbage collection, then you really start to up the level of programming styles.

You can write in a style where you have much more complex function results, and you don't have to worry about who de-allocates this. Who owns this memory. Who's supposed to get rid of it again, and you don't have all of these cycles that you maybe get if you implement with Ref counting or whatever you do to keep track of your objects. So, it made the world a lot simpler in a lot of ways. Then, the fact that these languages can be memory-safe, that removes a whole bunch of other. The thing that saved us all at the time was that there were just only so much capacity. There's only so much you can do in K.

Even though Biulds and 64 was going to be enough, there's only so much you can do on 640 also. So, that always saved us. It was like the complexity was never big enough that you couldn't get away with it. But once you're in this now, is literally a bottomless pit.

There's so much code, and you take so many dependencies, and if you don't have some of these guarantees built into the system, you're going to be spending an awful lot of time just chasing down dumb bugs that really you shouldn't. That's not a creative process. That is just a waste of time.

**KEVIN SCOTT:** Yeah. So, I want to get to that, but before we jump into modern type, let's talk a little bit about Delphi. So, maybe the compiler didn't change all that much but my god, that would be-

**ANDERS HEJLSBERG:** That was a revolution on the IDE side. It's interesting because I've worked on a project in between. I never went anywhere. It was code-named Monet at Borland, and it was about building applications visually, which was very sort of- Back then everyone was talking about software ICs, and you could just plug and play and draw lines between the inputs and outputs on the different things.

We were building this very advanced set of components that you could just plunk down on a canvas and wire together, and making Fahrenheit to centigrade converters was super easy. But it turned out that making big apps was horrible because you had lines going from everything to everything, and it just didn't scale. But I think in that process, I got a real appreciation for what does work well visually.

Like designing input and GUIs and whatever, and GUI was starting to happen. So, there was definitely a need there. So, by the time this Monet thing did not pan out, we realized it wasn't going to pan out, then we went back to basics. Then go, "Okay. Well, let's actually get into the GUI era with this product." Then there was this thing called Visual Basic out.

**KEVIN SCOTT:** Correct.

**ANDERS HEJLSBERG:** That taught us a lot, but also had a lot of issues. Like it didn't have object orientation, it was interpreter-based, it was generally frowned upon language at least by a lot of people, and so, we go, "We can do this. We can do this, and we can do it better."

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:** That was the genesis for Delphi. Then I think in the process of building Delphi, we also realized that- and there is this I think was Zach Urbach, our program manager who astutely realized that you can't just ship a Rad tool. You've got to have some angle, and the angle at the time was client-server. So, we actually pivoted it into a client-server tool. That gave us an inroad into enterprises and companies that were really building real database connected apps. I remember competing with PowerBuilder. I don't know if you were with PowerBuilder. So, that gave us a unique vantage point compared to Visual Basic.

**KEVIN SCOTT:** It was sort of an interesting thing. Delphi came out shortly after I graduated from college, and I became a real software engineer, I remember being so grateful for it because I did a bunch of work with it that clients were paying me for, and the tool made me look better than I actually was. We had made this transition from, you build in these text-based apps where if you really fancy, you're sort of blasted ASCII codes and the screen buffers. But Windows 95 came out. The onus was on you. It looks slick. It was a pretty Operating System.

**ANDERS HEJLSBERG:** It was an awesome way to build client apps. It really was, and then it had a bunch of fun things in it. I remember the two-way code generator. You can either visually design your form or you could go in and modify the code and then the form would change, right? Making that work was not easy, but once it did work, oh my God, it was so cool.

**KEVIN SCOTT:** That must be a gratifying thing. You've been doing this your entire career, but in my mind, a force multiplying effect of building developer tools as you put your development effort into this thing, and then it gets into the hands of the huge number of people.

**ANDERS HEJLSBERG:** That is the thing that I think in retrospect has been hugely rewarding. It's like when you build end user apps, you don't build the same kind of fandom as you do when you're building developer tools. Because developers, they put in the hours. Do you know what I mean?

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:** They come to really depend on their tool. They come to truly appreciate what they do.

**KEVIN SCOTT:** Yes.

**ANDERS HEJLSBERG:** Then they become these ardent fans.

**KEVIN SCOTT:** They are almost religious, right?

**ANDERS HEJLSBERG:** Yeah, I know, and that is so rewarding. Throughout my career, that has been so rewarding to go speak at a conference and have all these people come up and tell me how you saved my life. You did this. People have literally told men that. It's like guys who were in Russia, and couldn't put bread on the table learned to program with Turbo Pascal and literally this guy is claiming I saved his family's life. I mean, that I'm just God. That's amazing. That is wonderful.

**KEVIN SCOTT:** Really great.

**ANDERS HEJLSBERG:** Yeah.

**KEVIN SCOTT:** So, at the time you sort of referenced it already, there was this intense competition amongst commercial entities, companies like Borland and Microsoft, and the guys who were building PowerBuilder. There were these database language things going around and literally catalogs full of shrink-wrapped software that was just for developers to help them get their job done. So, the big thing that came on the scene right after Delphi was Java. How did that factor into your move from Borland to Microsoft? Because, in a way C-Sharp, it seems like [inaudible].

**ANDERS HEJLSBERG:** There is an arch there definitely. But I do remember, I think it was in ' when Java made its first appearance. This is where the Internet was starting to happen. Particularly, I think in the development tool space and programming, there was this collective madness around Java where everyone thought it was over. It was done. There were going to be no more [inaudible] and Java to take over the whole world.

There was a Java fund. Remember the Java fund? Funded by hundreds of millions of dollars whose sole purpose was to just invest in companies that were building their software in Java. As if that would make a difference. It was brilliant, what Sun did, I think that even they had no idea what they had unleashed here, right? But it was discouraging in many ways because we felt that, yes, that's fantastic for building applets in the browser. But you can't build real Apps with this thing.

But hey, you know what? There were the shiny object. There was collective craziness over the Java thing, but there were also things that I thought were interesting. This whole notion of write once and run everywhere is even today, it just so happened that it wasn't actually born out by Java. It was born out by Javascript. But, which was there already in the browser and ignored for a decade. It's crazy.

We'd had a JBuilder tool we built at Borland, which actually was built in Delphi. They started with the Delphi IDE, chopped it down, and then the language service, well- They weren't really called Language Services at the time, but the Mini compiler analyzer are used by the IDE to do all of their syntactic highlighting and whatever that was also written in Delphi. So, I got to know Java as a language. Then at the same time, there was a change afoot at Borland. Borland had a bit of an identity crisis, in the sense that everyone was up leveling how business was done.

It all became a game of platforms. Oracle was databases. Sun was workstations. Microsoft was the OS, and then Office became another platform. And Borland never really got the platform up leveling. It was always a bunch of separate products, and we try to bundle them together, but it never really made sense.

Then the company was so singularly focused on trying to get a foothold in that business that they started neglecting their developer tools, which was really where the company came from, and a series of management revolving door. Then I got more and more disillusioned. Then of course, Borland ended up going through a bunch of reductions in force if you will, and we spread a bunch of people into the universe.

A bunch of them went to the valley. A lot of them went to Microsoft. Then of course, they started calling, every summer when the weather is nice in Seattle, I'd get a call from Brad Silverberg, who had run R&D at Borland, and now was running Windows development. So, I would come up and we'd talk, and the first year I wasn't interested. But then eventually, it was time. So, I joined Microsoft in '96.

**KEVIN SCOTT:** Was C# the first thing that you worked on?

**ANDERS HEJLSBERG:** No. Not at all. There was no C# There were.Net at that time. I joined to work on Microsoft's Java development tool set.

**KEVIN SCOTT:** Interesting.

**ANDERS HEJLSBERG:** Yes, and I've worked on Visual J++. So, at the time, we had a Java development tool, and it was really just C++ IDE with the C++ compiler taken out and a Java compiler stuck in there, right?

**KEVIN SCOTT:** Yes.

**ANDERS HEJLSBERG:** That was visual J++ 1.1. I think. Then, we jumped from 1.1 to 6.0 because we realigned all of our Rad tools, VB was at version 6, and so, it became Visual J++ 6.0, and it had a Visual designer. You could design applets and whatever. But then we were also doing stuff that made it a better language for writing Windows Apps, and that ended up being very controversial.

I have personally read the contract we had with Sun. It was explicitly permitted in there that as long as we ran all the standard tests, we were free to do additional inventions. But that blew up. Literally, within three months of our shipping Visual J++, our product was enjoined by a judge in San Jose, and we were required to put in a warning dialog, "Warning! You are about to turn on Microsoft proprietary extensions. Are you certain your wish to proceed?" And that was just so you could use com interop. That was considered so evil. I was like this is crazy.

Inside Microsoft at the time, the Java fever, it was all over the industry and there were literally fractions of groups at Microsoft that believed that it was all going to be Java. We should build a Java OS, it should all just be Java. But then that happened the whole Sun lawsuit and whatever and that was sort of the genesis for us understanding that, listen, it's really hard to do what's right for your customers by extending someone else's platform. We've got to build our own that works for what it is that our customers are telling us that they want. That was the genesis of dot net and of course, we needed a programming language. I so happened to find myself at the right time at the right place for that.

**KEVIN SCOTT:** So how fun was that? Is that the first time that you had a blank slate?

**ANDERS HEJLSBERG:** Totally. Yeah. I mean can you imagine the opportunity to have a company like Microsoft put their might behind a programming language that you're going to create. I was flabbergasted that I was given that opportunity. Yeah, sure I'd proven myself but there were still a lot of other people at Microsoft who had different ideas about what should be done and then ultimately I was the one who was handed the task. So that was wonderful and we had a great team on that early C sharp compiler which originally was written in C++ or C plus minus because we didn't use all the C++.

**KEVIN SCOTT:** Which is a whole other discussion.

**ANDERS HEJLSBERG:** Yeah, yeah, I remember a guy named Peter Goldie wrote the core implementation of the compiler, I wrote the language spec. We had a wonderful design team process that we set up that you know to this day still exists. Where we used to have three meetings a week for two hours, where we would just do the language design. Iterate on it, you know what I mean? And Scott Wilson with our program manager that I worked with for almost a decade was the PM. So for four years he commuted from Hawaii. We would do our design meetings on speakerphone with him. It was a great time.

**KEVIN SCOTT:** Based on some of the stuff that you've talked about already and just sort of looking at C sharp and then typescript, you're fairly pragmatic when it comes to language design. You're not one of these language designers is like, "Oh, let's throw a Haylie Milnor type system and-"

**ANDERS HEJLSBERG:** I've always been a believer in understanding the problem and feeling the utility of something before I want to go implement it. I have to believe in it. I don't just throw it in there, yeah, I need a type system, want an Haylie Milnor or whatever. Now, no. I want to understand why is it there and does it really need to be there?

**KEVIN SCOTT:** And a real programmer is going to benefit from it.

**ANDERS HEJLSBERG:** Exactly and so that was the case in C sharp and that drove a lot of our decisions. For example, we added properties as a first class language construct because everyone was doing properties. It was like that was how programs were written and Java didn't have those, for example. So there were pragmatic decisions that we made.

So whenever someone goes, why is this feature there? It's not just because I thought it was fun, it's because it solves this, this and this problem. And ideally, with every language feature, this is something you come to learn over time is, people will request a lot of features but they will always have synthesized their idea of what it is that they want. They really come to you with an instance of a class of problem and they tell you I need this instance. And then-.

**KEVIN SCOTT:** So the trick is to get them to really describe the problem.

**ANDERS HEJLSBERG:** No, the trick is for you to understand what is the class of problem.

**KEVIN SCOTT:** Got you.

**ANDERS HEJLSBERG:** And that's what you go implement. So I never wanted to implement a language feature unless I could see multiple good uses for it. We're not just going to implement because it does this one little corner of XML well and that's it. I want this to be a generally useful thing that you can do other things with as well.

**KEVIN SCOTT:** Because it's a very serious thing to bake something into a language.

**ANDERS HEJLSBERG:** The thing you come to learn about languages is you can add all you want but you can never take anything away. At least if you care about backwards compatibility. And we've seen lots of train wrecks where people did not care.

Like say the transition from Python two to three even when you're just doing it in the name of cleaning up stuff that isn't right or isn't ideal or whatever. You can't do that to your user base. They have too much of an investment, it is your responsibility to ensure that their code continues to compile and work and then you can add more on top. And if you're not going to do that then you might as well blow it all up and go create a whole new thing that advances the state of the art. The worst thing you can do is the 99% compatible, that is absolutely the worst thing.

**KEVIN SCOTT:** So some of that attachment to the old is emotional but you can very quickly have hundreds of millions of lines of code across.

**ANDERS HEJLSBERG:** If you're a serving enterprise, well, it doesn't even matter enterprise is like anyone who writes a large body of code and then gets adopted somewhere, does not want to have to spend a bunch of time changing it in random ways just so it's prettier. No, big code bases get ugly, they all do. Then programming languages do too. I sometimes joke that show me the perfect programming language and I'll show you a language with no users, because only then can you go change it.

**KEVIN SCOTT:** I did an internship at Cray Research when I was a younger engineer still in grad school. One of the most mind blowing things that I had ever heard up to that point was I was sitting at lunch one day, one of these Cray veteran engineers was describing this sales process. So, they were trying to sell a supercomputer at some point to a car manufacturer and because they had this big 4 train codebase, it was due in finite element analysis. So they could simulate crashes on these supercomputers.

They'd come out with this fancy new version of unicode, so it was just going to be impossible for the car company to rewrite their 4 train code base which was tens and tens of millions of lines of code to work on this operating system for this new machine and the guy's looking at it like, all right, this machine is millions of dollars and that's tens of millions of lines of code. The operating system kernel is 25,000 lines of code, we'll just write a custom operating system for this.

Being able to unpack where the abstraction and complexity is, is really sort of an interesting skill you have to develop as an engineer.

**ANDERS HEJLSBERG:** Yeah. So pragmatism was always a main driver for me and it continues to be.

**KEVIN SCOTT:** What lessons is a language designer or like as a builder of development tools, have you learned that you're applying the typescript right now?

**ANDERS HEJLSBERG:** Well, we touched on a couple of them. You really have to respect and value backwards compatibility. Doing language design, it's interesting, there's really in a sense two phases to designing a language. There's 1.0 and then there's everything else. And with 1.0 it's a greenfield. You can do whatever you want to do. It's crazy fun and you can be super inventive.

But then from there on out it becomes a very different discipline because now, it's a game of trying to sneak in new features in a way that does not break backwards compatibility. And that is very different. And also you got to be very cognizant of every programming language sort of has a gestalt to it, if it's an object doing a programming language, you can move it towards functional program but you can't make it into a functional program. Like where immutability is first. So there's-.

**KEVIN SCOTT:** There's iterators, but yeah.

**ANDERS HEJLSBERG:** Right, there are certain things where you just got to go, "No, if we do this that'll just give people two ways of doing the same thing and now there will forever be confusion about which one it is that I'm supposed to use. So, we can't do that." So, after 1.0 you have to learn when to say no and save it for the next one, which is hard because there are things you want to do but this is not the right place to do it. And you could sink the ship if you did and then, I think you learn also to withstand the pressures of marketing.

I can't tell you how many mails I have written to defend our decision not to put XML literals into C sharp. I literally spend a year writing that email every two weeks. And Visual Basic did put XML there and I was always like "Listen, there are so many different ways you can write this markup. If we put it in an XML sinks then we sink along with it." Can you imagine if we'd had XML literals in C sharp. Now what-.

**KEVIN SCOTT:** I can't even imagine.

**ANDERS HEJLSBERG:** And it's sometimes you just got to stick to your guns and the face of adversity and go "No, this does not belong here, here is what I know, I understand how with the vantage point of today it might look like it does belong here but it doesn't belong here, long term."

**KEVIN SCOTT:** Yeah.

**ANDERS HEJLSBERG:** So, there are lots of interesting things that you don't realize up front.

**KEVIN SCOTT:** I've always thought that the better design programming languages were the ones where there was some coherent philosophy that was developer oriented up front. Pearl, which in many ways this is I've written a lot of bad Pearl code in my time. It's really easy to do and it's my fault, not language's fault but it had a lot of good stuff.

Like Larry Wall the creator of Pearl had these philosophical tenets like make the easy things easy and the hard things possible. I remember hearing Fred Brooks at one point say one of the best things you can do in language design is to figure out the things that you're not going to let people say. Because that's where mistakes happen. So it's always fascinating to me what these philosophical principles are because there's no one right way to design a programming language.

**ANDERS HEJLSBERG:** Right, but there are lots of wrong ways, and I think that one thing that people tend to forget too about programming languages is well a couple of things. First of all, it's fascinating to look at how slowly they evolve and how similar the programming languages of today are to the ones we use 50 years ago.

It's really quite frightening. I mean like Pascal was invented in 1974, so we're talking about like 40 some years here and it doesn't really look all that different. Okay, we've had a few, okay, object orientation but then Pascal looks a bit like a functional programming language, it doesn't have pattern matching, it doesn't have lambdas or whatever but it has some of it like functions and data being separated and anyway.

So that's one thing you come to realize. I think another one is that it's very easy to focus on one shiny thing that you're programming language is going to do. But then you always forget about the 90 something other percent that every programming language has to have. And that 90 other something percent keeps getting bigger and bigger now because people need tools, they need IDEs, they need statement completion, they need code navigation, they need blah, blah, blah.

And so whenever you're like, "Oh I got to have my little DSL like thingy here just for configuring this thing, oh wouldn't it be cool." Well, are you going to build an ID VoIP or you're going to build two? "Oh no, they're never going to get big enough for these programs." Well, every configuration file grows up to be its own crappy programming language eventually and if you don't watch out. And then they get bigger and bigger and bigger and now you need tools and now there we are repeating history again.

**KEVIN SCOTT:** Some of the hardest things that I've had to do as someone leading teams of engineers is to be the bad guy coming and saying, "Okay, we're not going to use that shiny new language."

And it's precisely for that reason when you're figuring out what development tools you're going to use to empower all of your engineers to build great things, you have to make choices because all of the apparatus that you build around them to support them, what's the build system going to look like when you've got hundreds of millions of lines of code and thousands of engineers? How do you do continuous integration and continuous deployment? What is your testing regime look like?

So you build all this automation that sits around the programming language, the engineers can sometimes have this notion in their head it's like, "Oh, it can't be that hard, I'll just write my one thing in this and it'll be okay." And then-

**ANDERS HEJLSBERG:** That can happen to you, it particularly the higher you rise in the engineering ranks and the less code you write, the more possible you think the impossible is, right? You become an architecture astronaut where surely, anything can be done here, right? Look at this diagram, I just drew it for you. Just go and make it so, right? No, coding is hard and it continues to be hard. Code gets bigger and bigger and bigger but our brains are not getting any bigger, and this is largely a brain exercise.

**KEVIN SCOTT:** Yeah. So one of the things that I end up talking with fellow engineers about a lot over the past several years is where we are now as a discipline, as a profession, compared to where we were before.

So, in some ways, the programming languages from a language perspective aren't that much different now than they were. But you do have these crazy things like cloud computing and open source software where, I can't tell you how many times, when I was a young engineer I re-implemented chain hashing implementation. It's like I could probably still write the code just out of finger memory.

**ANDERS HEJLSBERG:** Right. Right.

**KEVIN SCOTT:** But like so much of this stuff is accounted for you, so, you've got really robust SDK's and like a standard class library for programming languages and then you've got like all of this infrastructure. What you can sit down in a weekend and write in terms of internet service application, compile it, press long button and ship it to a cloud, then expose it to the world. It's like crazy.

**ANDERS HEJLSBERG:** No. It's phenomenal what's happened here. I think a lot of it is like- opensource has done a tremendous amount to shorten the distance between individual programmers and between users of someone implementing a tool and someone using a tool. We're this close. I mean, I feel it on my body every day, right? I sit on the GitHub issue tracker on TypeScript and then I'm talking to the very people who are using it and if they report a bug, we can have it turned around and have it in the nightly bill that evening, which that used to be a two-year cycle. We have done it like old proprietary style. So, the pace of advance there and iteration is just gone up so much. That is-

**KEVIN SCOTT:** It's also that you doing it in the open, right?

**ANDERS HEJLSBERG:** Sure.

**KEVIN SCOTT:** Like so many people are making so much of this stuff available for low or no cost.

**ANDERS HEJLSBERG:** Yeah. That is still a strange phenomenon, because ultimately, someone's got to pay, right?

**KEVIN SCOTT:** Right.

**ANDERS HEJLSBERG:** I don't know where that's going all land eventually, but I have to say I do love the process of doing it. It's sort of like a return to the things that really matter about programming. I mean our team, it's not a big team but it allows us to be craftsmen and it allows us to really just continue to iterate.

I was always a great fan of- you don't just build a product and throw it out there and then you move on to a whole new thing. No, you got to come back and polish it. Then iterate on it and make it better and make it better. It doesn't get beautiful just out of the box. It gets beautiful over a period of years of constant iteration and care. Do you know what I mean?

**KEVIN SCOTT:** Yes.

**ANDERS HEJLSBERG:** That's something that this ecosystem allows us to do at such a phenomenal pace that it's just beautiful. We could have never done what we're doing now with Visual Studio Code and TypeScript in the old proprietary world. It simply would have taken 100 years to get that amount of feedback incorporated on the kinds of cycles that we were on back there.

**KEVIN SCOTT:** Right.

**ANDERS HEJLSBERG:** So that's great.

**KEVIN SCOTT:** That's awesome.

**ANDERS HEJLSBERG:** Yeah.

**KEVIN SCOTT/ANDERS HEJLSBERG:** I think maybe that's a good point to stop. We've gone from core memories to the sort of near infinite rate of change in open source software. So, thank you so much.

**ANDERS HEJLSBERG:** All my pleasure. It was great, fun.

**KEVIN SCOTT:** It was awesome, so fun.

**ANDERS HEJLSBERG:** Fun talk, yeah.

**[MUSIC]**

**KEVIN SCOTT:** Thanks for joining me from Behind the Tech. It was a lot of fun speaking with Anders. I had some surprises in there, like I didn't realize that he had written a precursor to Turbo Pascal while he was still in university.

That may very well be the first integrated development environment in the entire world. And he's so modest about it, like, "I never thought of it." It's like God Almighty, this is one of the more important breakthroughs from point of view of the developer that has happened in the history of computing. So, many thanks to Anders for being on the program.

**[MUSIC]**

**KEVIN SCOTT:** Next time on Behind the Tech, we'll hear from Alice Steinglass who heads Code.org. We'll hear about how she's working to make sure every kid has the opportunity to take computer science classes in school. Be sure to tell your friends about our new podcast, Behind the Tech, and to subscribe. See you next time.