Simon Peyton Jones

Interviewed by

Ian Symonds

19 November 2018

At the

Microsoft Research Centre, Cambridge

Welcome to the Archives of Information Technology. It's the 19th of November 2018, and we're at the Microsoft Research Centre in Cambridge. I'm Ian Symonds, and I've been working in information technology and management consultancy since 1976, a period of enormous change in the industry. Today I'm talking to Professor Simon Peyton Jones.

Simon achieved a First Class Honours at Cambridge, and has since dedicated his career to functional programming, and was the designer of the Haskell functional programming language. In 1989 he became Professor of Computing Science at Glasgow University, and is now a principal researcher at Microsoft, and retains an honorary professorship at Glasgow. He also, last year, became a Fellow of the Royal Society. We'll be talking about Simon's background influences, some key events that shaped his career, and his views on the industry today.

[01:11]
Simon, can we start by asking where and when you were born?

Yes, I was born in, in Simon's Town, in South Africa. That's why I'm called Simon apparently. My dad was in the Navy at the time, when there was a British naval base there. But we came back to England when I was quite little, just a few months old. Lived in England for a few years, before moving to Trinidad, where I spent the time between when I was about three and when I was about eight.

OK. So did you, as a, as a young person, did you move around the world quite a lot over time?

No, that was it. Because Dad had retired from the Navy by the time we got to Trinidad. He was running the Trinidad and Tobago Coast Guard. So really, it was Trinidad. And then, once we moved back to England, it was fairly much in one place.

Then he had a career in the Duke of Edinburgh's Award…

That's right. When we came back from Trinidad, he started, he took over as the director of the overseas side of the Duke of Edinburgh's Award scheme. And that served him for the rest of his professional life.

So he went, he just travelled, travelled there and back…

He travelled a lot. We stayed in Marlow, yes. [laughs]

[02:15]
OK. So what was your family life like?

Oh well I had three brothers and sisters, so there was… All fairly close in age. So, I guess that dominated my existence, you know. We were quite a close family, and are still, I still, I'm happy to say that we are still quite a close family, rather a long time later.

Yeah, you're all, you were all close together in age, weren't you?

Quite close together, about, you know, a year and a bit apart, mostly.

Yes.

Yes.

Yup.

[02:42]
And what were the important influences on you in your early life?

Oh well, clearly my family and my parents. I think, I mean school was, clearly, a big influence. I mean I went to boarding school when I was eight, so that was a, that's something that, you know, it's not, it's not something that I would seek for my children, but it clearly had a big influence on me. At the time my parents were thinking about going to work abroad again. In fact that didn't happen. But I ended up

staying at boarding school. And I cannot think but that must have a major effect on your whole, you know, life trajectory, if you do that. I remember it as being a, you know, a happy time. But clearly being separated from your parents at age eight for quite a large fraction of the year cannot but be significant.

Yes. Did you feel you suffered in, in that way, you know, emotionally, from that?

Not… I mean I don't… I don't remember any feelings of suffering. In fact I remember quite a lot of feelings of opportunity. Because a boarding school is a, it's kind of like a little bubble, you know, you exist within it. The entire world outside, you know, you just about hear if man has landed on the Moon, but pretty much nothing else penetrates the bubble. It's a whole little world unto itself. With, you know, a lot of close friendships, and a lot of opportunities that I enjoyed very much, including when I was at Marlborough, my first introduction to computing and electronics and all of that.

[04:05]
Mm. We'll come on to that in a moment. As I understand it, your father and your grandfather were both navy men.

They were.

Was there any pressure on you to follow in their footsteps?

Oh well, yes. In fact while at Marlborough I had a naval scholarship, towards the fees. So I did apply for a naval scholarship. And at that stage I thought, maybe I'll, you know, do something like, I was engineering focussed, I think maybe I'll become an engineer in the Navy. But then, very helpfully, I went for a week's sort of, training course at Dartmouth, and came away realising the Navy was absolutely not for me. I have a huge respect for people who work in the armed services, but it was quite clearly not suitable for a slightly geeky [laughs] young man as I was then.

[04:47]

Could you trace your education for us, through early schooling, secondary schooling, college and higher education? Which, you know, which schools, colleges and universities did you attend?

Yes. So I went to this boarding school age eight, it was called Lambrook. It was a small prep school with 130-odd pupils. That was I guess eight till, thirteen-ish. Then I went to Marlborough College, a public school in Wiltshire, which was much, much bigger, hugely influential on me I think. And then… Then, while I was there, I resisted for a long time the idea that I would go to Oxford or Cambridge, but in the end I yielded, and, I came for interview at Cambridge and loved it. And I applied for a place. Did a gap year, I had a gap year, but, spent three years at Cambridge doing an undergraduate degree, then one year postgraduate diploma in computer science. And that was the end of my formal education. I did not do a PhD for example.

What did you do in your gap year?

What did I do in my gap year? I worked for IBM, actually. So I did not go abroad. I worked for IBM in Croydon, and that was my first exposure to corporate IT, and it was a, it was a huge eye-opener.

But I understand, when you were at Marlborough College you were keen on IT and programming even then. You said you programmed the school's computer.

Oh yes. So that the first thing. At the time, the, the first microprocessors were just coming out while I was at school, the Intel 4004, and 8008. So, we didn't have access to any microprocessors, but we did have an electronics lab, and we did have a strange thing called an IBM schools computer, which was my first programming experience. It had 100 memory locations, a lot of programming, which could only be programmed in, in, with numbers, you know, no high level languages at all. Just type numbers into the thing. Didn't have any long-term memory, you had to type in your entire programme from scratch every time you switched it on. And also we had access to a, I took the bus to Swindon where the local technical college had an Elliott 803 computer that filled a whole room. So that was my early introduction to computing,

which only a handful of students were involved in. It wasn't a taught part of the curriculum at all.

And what were your main subjects that you studied at Marlborough College?

Oh, well, up to, up to GCSE, what was O Level, no, a wide variety of subjects. I don't know, maybe I took twelve or fifteen O Levels, I forget. But for A Levels I did maths, physics and chemistry, and further maths.

The standard triple. Yes.

Yes, sort of, standard triple sort of science focused, but quite a lot of extracurricular activities in astronomy and electronics and such like.

[07:18]
OK. Good. So then you went to Trinity College. What subject did you study for your degree there?

Oh, so I… So I entered as a mathematician. So, at that age you couldn't do a computer science degree. I was pretty sure that computing was where I was going to end up, but it was my hobby. And so in any case I sort of thought, well, it's a bit, it's a bit cheating to do your hobby as a university degree. Not that you could anyway, except as your, a final year course. So in the event I elected to do mathematics, and, was lucky enough to get into Trinity, where I, it was a huge shock to the system, because I thought I was quite a good mathematician, but actually, you know, Trinity mathematicians are a kind of breed apart, and, they were just streets ahead of me. I was in shock for the whole of the first year. And in fact I, in the end I, I stopped maths after two years, because, while I was, sort of keeping up OK, I could see that my contemporaries, they were like ducks swimming on the surface of the water enjoying themselves, and I was struggling just to get a breath every now and again. So in the event I, I did enjoy maths a lot, but I switched to electrical sciences, which is a final year course, run by the engineering department. It was populated almost entirely by refugee mathematicians and physicists. Because the engineers considered it to be too difficult, but the mathematicians and physicists considered it to be

relatively easy. So we all, we all migrated into there, and there was a happy band of 20 or 30 of us who took the final year, one-year final year course called the Electrical Sciences Tripos.

[08:50]
And, I'm sure you look back on that time with some pleasure. What was your life like then, working, studying and, and at play at Cambridge?

Yes. Well I, I enjoyed Cambridge hugely, hugely. Made a lot of friends. Worked pretty hard, so there was a lot of, I had a lot of lectures, so, you know, very morning, nine till one, was really pretty much all lectures with an occasional spare slot. I did a lot of work between, you know, 11 p.m. and 2 a.m. I went out a lot in the evenings. I did quite a lot of… not, not so much… No, it was kind of, socialising with people in other colleges. Not much clubbing, there wasn't much clubbing in those days. [laughs] In fact none at all. It was the age of when the students were agitating about apartheid in South Africa. And, and quite a bit of stuff with friends of mine who were interested in computers, building, you know, we were busy building and programming computers, including late at night.

[09:55]
OK. And it was at university that you first encountered functional programming, as I understand it, is that right?

It was, yes. That was when I met Arthur Norman. So he's a, still, a professor in the computer science department here at Cambridge. And he was, he gave a very shot series of about four lectures on functional programming, in which he showed us programmes that I thought were just, you know, could not be written, like, how do you build a doubly-linked list without using assignment statements, to, you know, connect up the blobs? And, he was… He had come through the Lisp tradition, so he was quite an experienced functional programmer, and he sort of, opened my eyes to the idea of functional programming. And there was a little group of us, John Hughes and Thomas Clarke were others, William Stoye, Jon Fairbairn, who… And so there was a little group of us who became literally inspired by the idea of functional programming as, and, you know, we'll talk more about it, but a whole new approach

to the very enterprise of writing programs at all. And so that was very influential on me.

[10:53]
Just, perhaps you could just explain, this might be an opportune moment for you to kind of just explain what the principles of functional programming are, up to a high level.

Yes. Yes. So, I think the, the easiest way to get at functional programming is to think of spreadsheets, right. So a spreadsheet, in a spreadsheet you write formulae which you evaluate to get a result. And these formulae of course do not have side-effects. It wouldn't make sense to say, the value of cell A1 equals print 'Hello' plus three. I mean, when would print 'Hello' be printed? Well it depends when the spreadsheet decides to evaluate the cell A1, and that's by design not an issue that the programmer should even have to think about. So, in spreadsheets, we instinctively programme without using side-effects, without calling functions or procedures that do things other than return a result. So, the surprising thing though is that, that programming paradigm, programming using only pure functions that take input and produce results but have no side-effects, that programming paradigm is not just, it's not just possible to write programs that way, it's actually the foundation of an entire, you know, entire computational paradigm. If you think about, Turing machines are, if you like, a definition of what it means to be computable, what can we compute, that's defined by a Turing machine. But that's a very imperative thing, modify a tape by writing symbol, reading and writing symbols. A similar time that Turing machines were being invented by Turing, Church, his own supervisor actually, was working on the lambda calculus, which is a much more functional one, an entirely functional approach to, thinking about what it means to be computable, right? And it turned out that these two approaches were in fact interdefinable. So, if you can compute it with a Turing machine, you can compute it with a lambda calculus, and vice versa. But, you can, this is rather reconstructing history, but we could look back and say, it is as if functional programming is built on lambda calculus, and imperative programming, mainstream programming, is built on Turing machines. And so the, the revelation for me was that, functional programming was, you know, an entirely radical and elegant attack on the entire enterprise of programming. And that's rather attractive to a, to an

undergraduate, who thinks, you know, who wants to reform the world, to say, let's, we'll have no truck with the, you know, all the compromises that imperative programming entails; let's try this radical and elegant approach and see where it takes us. And it's taken us quite a long way.

[13:17]
And I gather that there was… Well you, you also talk about lazy functional programming, which is a, a specific flavour of functional programming.

Yes.

Just, can you say briefly what that is?

Yes. So lazy functional programming… When you call a function, in C or Java, you would typically evaluate the arguments of the function before you make the call. In lazy functional programming, you would instead, not evaluate the arguments; instead you pass to the function a kind of heap allocated recipe which when poked on will evaluate the argument. So, it's if you like the politics of procrastination, don't do the work until you have to. So, it's kind of like a little tactic that, it maybe means you do less work, but you also have higher overheads as you go along. But more interestingly, it turns out that it's a, it gives you a new form of modularity. Since the, since the arguments are not evaluated until they're needed, that opens up new forms of expression. For instance, you can write 'if' as a function. If you wrote a function called 'if' in C, well it would evaluate both the 'then' branch and the 'else' branch, before it decided which to, which it needed. With lazy evaluation, you just pass them both and only one of the two gets evaluated. And of course that's crucial to making 'if' work. So, it gives you access to a new form of modularity. So it was… And at the time, it was very, kind of cool, and it linked in a very principled way to something called normal order of evaluation for lambda calculus, so it was very, it had a very deeply principled story, and it connected very much with a class of implementations that David Turner was popularising called SK combinators, and he built his languages SASL and Miranda on top of those, and KRC. And those were all inspirations to that little subgroup of undergraduates as well.

[14:58]
OK. So, I mean, you said you had this sort of, very sort of, group, this group of fellow students and teachers who kind of worked together on this. So I, I imagine you must have had pretty good relationships with your, with your teachers and tutors? Were they, were they good?

Yes. Yes, so I think the… Well, I mean, they were amazing people. So David Wheeler, you know, was the guy who invented the subroutine. [laughs] I mean he wasn't really a very good lecturer, but it was amazing to be in the room with the person who had been right there at the foundations of computer science. Arthur Norman, a much better teacher, and as I say, still, still active today, was hugely inspirational to me. But in fact, we didn't interact on a daily basis at all. He sort of, set us off. One thing that was quite important at the time was that, John Backus gave his Turing Award Lecture. I think this was 1977. So, you'll know, the Turing Award is like the Nobel Prize for computer science. And he gave this lecture that said, whose title was, 'Can Programming Be Liberated From the von Neumann style?' in which he, who was essentially a god in the firmament, you know, we were worms as undergraduates and he was essentially at the at the apex of that pyramid that we were at the bottom layer of, he was saying, functional programming is the way of the future. So that kind of, was a very high level and senior endorsement of this, you know, stuff that was in the air that we, we found so inspirational. So it was very, it was very… It was great to have him saying, look, this is a sensible thing to study. It actually has… It's not just cool, it actually could be useful.

[16:44]
Apart from computers, did you have any other student age enthusiasms or interests?

Let's see. Some. Some. I wasn't really… I… But… Yes, not… We did a lot of punting, and, I would sort of, go out with my friends quite a bit. But… But I wouldn't say I had another… Oh, I joined the university air squadron. That was a, that was a… That was a big thing at the time, that took up quite a lot of time. And that, at the time you could join the university air squadron, there were teachers who fly. You didn't have to promise to join the Air force. It was an amazing… It was like, who would not want to be taught to fly at the Government's expense? So, I

indeed learnt to fly single-engine Bulldog planes, and flew solo, and that was great. And that was quite a big consumer of my time. But actually, by the time I had done all the sort of, extracurricular computing stuff, and this quite intense course, and the university air squadron, there wasn't a lot of time for anything else. [laughs]

OK.

Music, bits of music certainly.

Mm. What sort of music?

Singing. But…

Mm. OK.

Choirs and such like. Yup.

[17:52]
And, well, the next question I was going to ask, but I think you've probably answered this already, was, who and what were the most important influences on you at this time? But I, I guess…

We talked about that a bit. I mentioned David Turner.

Yes.

I think that he was very… He wrote these, a series of short papers about SK combinators, which had a, a very decisive impact on my sort of, early thinking about functional programming. And I would say also my peers. John Hughes, Thomas Clarke, later Phil Wadler, but not at this, you know, not at this time at university. Jon Fairbairn, Stuart Wray. These were, these were very clever, you know, all young men as it happened, and very influential on me.

Mhm. And were there any particular events that shaped your, shaped you during your education?

[pause] I don't think I can think of a particular moment.

Any turning points?

No.

[18:52]
No. No. OK. Have you… I mean, these friendships that you formed during your education, I understand, or I guess that, they followed you through life, have they, and you still collaborate with each these people?

They have. I mean, some more than others. But I went, only a month ago I was at John Hughes's sixtieth birthday festschrift in Chalmers. He since moved to Sweden. And I've collaborated with him on and off through my professional life. And, and yes, I mean, my other colleagues from Glasgow, I've certainly collaborated with over an extended period.

Mm.

I'm… I'm the sort of person who… Some people think best when they're in an isolated room and they can just be quiet and think. And I'm almost incapable of thinking under those circumstances. I rapidly go into a, a sort of, loop. I can… The way I think is by sitting with a whiteboard with somebody else, in dialogue. So, for me, collaborators who are prepared to, you know, accompany me on this sort of intellectual journey are hugely, hugely important, because they're the means by which I think.

[20:02]
OK, that's, that's interesting. What were the key lessons you learnt from your education that, or that, that… What lessons… To what lessons do you attribute your later success?

[pause] Well I think one of thing that I did absorb from my education was a, you know, a love of learning. It is actually exciting. There's just… Mathematics and computer science is just full of amazing, clever, elegant, beautiful, satisfying ideas, and that's, it's an adventure to discover all of those things, and then, maybe later to be able to contribute to them. So… And, also, I think that perhaps, encouragement, and maybe even permission, to, spend sustained attention on trying to make things simpler, trying to identify the beautiful essence of something, rather than just, you know, pile in there and make it work somehow, keep throwing mud at the wall until enough sticks, you know. [laughs] That's a, that's a, a privilege that researchers at universities have, this really applies to the rest of my life, to be allowed to spend time, not just throwing mud at the wall, but to say, what makes mud sticky? [laughs] And how could we make it stick better? You know, what's the, the essence? That's a huge privilege that academics have. They're not required to, you know, just deliver a product in the next, you know, six weeks or three months, which people who work in companies pretty much are.

[21:38]
Moving on to your career. I mean, you did actually spend a couple of years, kind of in industry, I suppose at, your first job was at Beale Electronic Systems.

That's right, I did. So, when I left university I had no notion that I would eve return. I was fully… I had no plans to become an academic when I left university. Indeed, the engineering department wrote, those of us who got first letters, saying, 'Would you like to do a PhD with us?' And, [laughs] I think we, we all replied saying, 'No. We're going to go and get jobs, thank you.' But… So I, I ended up working for a small electronics company. They built process control and monitoring equipment. And as is many things in life, it was entirely random. I just met Nicholas Beale, who was a charismatic and very effective entrepreneur, at a party. And we got talking, and before I knew it I was, I spent a summer working for him, before I graduated, and then went to work for him permanently after I did. It was a very small company, it was about, eight people. We were always bankrupt. It was… So it was quite, it was quite sort of, hand-to-mouth existence. I was the sort of principal software engineer really, and did quite a lot of hardware design as well, or hardware, debugging

certainly. So it was very, I mean it was very energising in some ways, but I also found it to be quite, quite daunting to be working for a company where I thought, if I'm, you know, if I'm late with the software, people don't get paid. So, after a while, after two years I found that a bit exhausting. And so I started looking around for a job.

[23:08]
And you returned to academia at that point?

That too was entirely random. Yes. My sister at that stage was studying at UCL. So I said to her, you know, I said, 'I'm just kind of looking around for other kinds of jobs. I don't really know what.' And she said, 'Well, you know, they are advertising a lectureship at UCL.' I had no idea that… I, it had not even occurred to me to look for a job in academia. But I thought, well, it can't do any harm. So I applied, just to see if I got an interview. And much to my astonishment, I was appointed. I had no, no PhD. I had… I did have one paper to my name, but, that was all. And I got a, you know, a permanent lectureship at UCL, which is a front rank computer science department. It was pretty amazing. I think in those days there as a, the computer science departments were expanding at a great rate, so, you know, if you had a pulse, they would hire you.

[23:56]
And, so, so this is where you started your research career, at UCL.

Yes.

Was that alongside some teaching and so on?

Oh yes. Yes.

Yes. And…

That was quite difficult, right. So… Because nobody taught me to do research. I had not done a PhD. So I would… You know, my head of department gave me a light

teaching load. He said, 'Well, you know, you need to get your research started, so I won't give you too much teaching.' So I, I would sit there with a sharp pencil and a blank sheet of paper, and wait for great ideas to happen. And, not unsurprisingly, nothing much occurred. So it took me a while to, and indeed some, some mentorship from others, from some of my colleagues in, and indeed from David Turner, to get started on something concrete.

[24:34]
You built this GRIP processor. Is that a graph reduction parallel?

Yes, GRIP was a, GRIP was a bit later in the day. So the first, I guess the first… I was UCL for about seven years. The first three or four years I think I was mainly, you know, just starting on functional programming research, and my colleague John Washbrook, who was a very important mentor for me, he said, 'Well, you're sort of flashing about a bit Simon. But instead of trying to do something grand and ambitious, why don't you just do something, right? No matter how, you know, insignificant it may seem. Just get started on something.' So I did. So I built little SK combinator, parser generators, and little compilers, and… They weren't very, you know, they weren't terribly exciting, but in the event, a couple of them did turn into papers. And it was very good advice. Don't be too ambitious. Just start with something modest and, make progress.

Get it under your belt.

That's right. So GRIP was a bit later. So when the Alvey Programme was being launched, it was a big funded research programme, in response to the Japanese Fifth Generation project, which was all about Prolog and logic programming, and expert systems. So, the British Government's response was the Alvey Programme. So I put in the grant to Alvey that said, let's build a parallel graph reduction machine. So graph reduction was our execution mechanism for functional programming. Functional programming is, one of its attractions is, it's inherently parallel, right. The very fabric of imperative programming is sequential. Do this and then do that. The fabric of functional programming is, is by design, parallel. So that was a very attractive possibility. So, I said, let's build this graph reduction machine, and they

gave us money to do it. And we did indeed build the thing. There's some of it in that box over there.

[26:20]
So how long were you at University College? It was seven years…

UCL was about seven years, yes.

Seven years, yes. And then…

And I became a lecturer and then a senior lecturer there.

And then, at that point, I think it was… You moved across to Glasgow University. Tell us a bit about that.

Yes. So Glasgow at that stage, Glasgow was… Glasgow University had, had a respectable computer science department, and decided to make it an outstanding one. And they did that by investing in the department. Universities are always making strategic investments, and, very often they, you know, they don't come to anything very much, but this occasion it really made a step change. It was an astonishingly successful investment. So, Malcolm Atkinson and Keith van Rijsbergen were leading this department. They hired John Hughes and Phil Wadler. And, and those two, with Keith and Malcolm, then invited me to apply for a chair they were advertising. And much to my complete astonishment, I was appointed as Professor of Computer Science at Glasgow. Then… So I… John was, John was the, appointed a professor at Glasgow, the youngest professor I know. I think I at that stage was something like 32, when I was appointed there. So I was tremendously excited. And it was a… It was…

That's still a great achievement at that age, yes.

It was a, well it was a… Well it was a huge opportunity really. They were explicitly saying, we're going to hire, you know, up-and-coming folk, rather than the established folk, and place a bet on them, which they were, you know. So I, I feel,

they, placed the bet on me, and I'm, [laughs] feel very grateful to have been the object of such a bet.

I suppose that was the great thing about IT at the time, you know, it was still expanding and developing.

Lots was happening, yes. Yes. And so they placed a fairly big bet across the whole department. There was a, you know, functional programming group spun up, and there was a very exciting functional programming group that was at Glasgow between about 1990 and 2000.

[28:22]
OK. And this… And, while you were in Glasgow, you, this is where Haskell emerged isn't it, as a functional programming language?

Yes. It was a little bit before that actually. It was more like, two or three years before that all… So at that stage…

So tell us something about how Haskell developed.

So, after I had started work at UCL, there were a number of other people elsewhere in the world, you know, John Hughes was one, but there were others, like Paul Hudak in the States, and Jo Fasel, and, and Dick Kieburtz, and David Wise and Dan Friedman. So there was quite a lot of people who were interested in lazy functional programming generally. But we all had different languages. And we realised that that wasn't very productive for us, right. So if we just… We thought, we'll try to do something very modest. Let's just agree a common syntax, right, so we don't gratuitously differ on our programmes. Because really we agree on the semantic core of our language. So, it was an extremely unambitious agenda. We thought, let's just spend a few months, you know, writing down a common syntax and we'll be done. Thirty years later, we're not done. So… But a group of us got together, and we started to design Haskell, and that was, I think, the first meeting was in about 1987 actually. We named it after Haskell Curry, who was a logician from the 1930s, got permission from his widow to use the, use his name. Haskell was his first name. And… And then by

the time we got to Glasgow, we had, we were, started to really get engaged on building the first compiler for Haskell. But it was a committee language, right, it was, furthermore, an international committee. So, three of us, myself, John Hughes and Phil Wadler, were at Glasgow, but the rest were scattered around the world, including in Sweden.

[30:04]
OK. And developing the compiler, how long did that take, and, you know, how did you go about that?

Well there were quite a lot of compilers for Haskell. Mark Jones was developing a compiler for Haskell. Thomas Johnsson and Lennart Augustsson already had a compiler for Haskell called HBC. Well they were developing at the same time. Ours was called the Glasgow Haskell Compiler, GHC. And, well, how long did it take to develop? It's now a, now at last 30 years old, and it's still developing. So it's not done.

How did you regard the other teams working on their own compilers? Were they competitors, or did you collaborate?

Oh, well, sort of. No. No, definitely a friendly competition, right?

Did you share, shared knowledge?

I mean the whole point was to have… Remember the whole point was to, not differ on syntax. But, maybe, you know, we didn't necessarily go on implementation technology when we go to use, you know, interpreting SK combinators, or compiler to machine code, or, supercombinators, or, who knows what, parafunctional programming, who knows? So, so we anticipated a big range of implementations with a common syntax. So… It wasn't at all antagonistic; rather, that we did regard, you know… Lennart and Thomas, the sort of, ace implementers, so we always regarded HBC as our, you know, the competition to beat for performance, when we were building GHC, yes. Lots of friendly competition.

[31:17]
OK. I think, as well as all this exciting technical work at Glasgow, you did have a sort of, like, management responsibilities there as well, didn't you, for…?

Yes. So I was, I was professor in the department, so I was teaching courses, I was, you know, on some committees, and chairing others. Latterly, from about 1985 to '88, I was the head of the planning unit. The university was divided into planning units, which are groups of departments. My planning unit was maths, computing and statistics. So I was, you know top of this group of three, well led and, you know, sparky departments. So, it was, that was quite a nice role, because they were well led. And I didn't have to, I didn't have to micromanage them at all. I would just divide up the money at the beginning of the year, and go to a lot of university level meetings and try to keep the crap off them. [laughs] You know, sort of, deal with the university interface.

Did you enjoy, did you enjoy that sort of thing, or did you prefer it when you were doing, doing the technical stuff?

No, much to my surprise I quite enjoyed it. I quite enjoyed it. So, I think the… I quite enjoyed it, because they were well led, you know, so that the… To create… Universities are exciting places to work, because they're very polychromatic, you're researcher, you are pastor, you are teacher, you are manager, you are mentor. All of these things together. And being a head of department or head of a planning unit is part of creating an environment in which your colleagues, who you love and cherish, can grow and flourish. And that's rewarding in its own way. It's very helpful being a limited term, right, so I knew it was just going to be a sort of, three-year term. And that means you think, OK, I can do that, and my research will take a back seat for a while, or, you know, less of a front seat anyway, and then I'll be able to get back to it. So, I think that's a strength of universities, that they often rotate these jobs. They're not regarded as, that's where you end up until you die. It's, you do it for a while as a service to your colleagues, and then you step down. And I had, in my department, you know, three previous heads of department to consult with and to advise me and prop me up when I was thinking about, going to, you know, some senate meeting, I

would then talk to Malcolm and Keith and say, [laughs] 'Guys, what should I say?' That was so helpful. And I think it's a great strength, of good departments.

[33:26]
And then, in 1998, September 1998, you moved to Microsoft Research.

Yup.

Can you just tell us how that came about, and…?

Oh that was also a bit random really. So, I had been a… I wasn't really…. I had been on sabbatical for a year, after seven years at Glasgow you get a year's sabbatical, and I spent that year in Oregon. So that was a very important year for me. And Portland…

Doing what in Oregon?

The Oregon Graduate Institute had a very good computer science department. Now, sadly, completely closed. They've essentially all moved to Portland State University. At the time it was a fantastic computer science department, and there was, John Hughes and Mark Jones – sorry, John Launchbury and Mark Jones were both there. And, it was them that I really went to work with. So I had a wonderful year in Oregon. Erik Meijer was there as well. And, when I came back, I was fully expecting to spend, you know, to continue at Glasgow, but at that time Dorothy, my wife, who is a priest in the Anglican Church, our children had started to get bigger, she was starting to look for more full-time work, but the Anglican Church in Scotland, which is called the Episcopal Church of Scotland, is quite small, and just doesn't have many jobs. So she said, 'Simon, can you find a job in England somewhere? I'm sure I'll be able to find a job nearby.' So, I looked around, and Microsoft Research, Microsoft Research was just starting in Cambridge, was, had been about, was about six or nine months old. Microsoft had decided to start a full-on research lab based in Cambridge. So I phoned them up and said, you know… Roger Needham was then in charge. [laughs] And so I said, 'Look, would you like to give me a job?' [laughs] They said, 'Oh very well.' It was a little bit more than that, you know, there were interviews and

things. But it was, they didn't headhunt me, because at that stage, MSR was very careful about not being seen to suck talent out of UK universities, or European universities generally. Instead, it was a more sort of, organic growth, people tended to, you know, apply, rather than be headhunted. They were quite careful about that. Because everyone was very anxious that we, that MSR would simply suck all the talent out of the European education system.

[35:29]
And how does Microsoft Research structure or, or… What's the strategy for the research it does here? Does it, does it… Is it driven bottom-up by the sorts of people that it recruits, like yourself, or, or does it have a sort of, a, an agenda or policy for, for where it's going?

That's changed quite a bit over time. So when I joined, it was, it was very much bottom-up driven. You know, the mission… I think the Microsoft Research's mission statement was, number one, push forward the boundaries of knowledge; number two, put Microsoft in a position to be agile when, you know, new stuff heaves over the horizon; and number three, provide a reservoir of expertise for, for the rest of the company to draw on. Nowadays the, the, it's more project driven, so we're more, the mission statement now is more, using deep research to have impact upon the world, and in particular on Microsoft's businesses. So that's, that's very motivating for computer scientists, because there's nothing that we want more than to see our work used in practice. But it is more focused. It used to be, one man and his workstation; now it's more project teams of researchers working together, usually closely aligned with product groups, to get some deep research into some production, into production essentially. So that's quite a big shift that's taken place over the last 20 years.

[36:55]
Mhm. And, what's Microsoft's interest in Haskell and the, and the compiler? Because you, you kind of brought that work with you, did you?

Yes. Yes, I brought it with me. So I was fully, I was fully, fully prepared for them to say, 'Well Simon, you know, we think you are great, and we think Haskell is very

nice, but it's just not for us,' right? Why would Microsoft want to fund you? Because I was pretty clear, that if they were going to hire me, it was because I was going to carry on doing this research, that's what I wanted to do. And, if they didn't want me to do that, well I would find a job somewhere else. But Roger said, 'No no, we're fine, that's what we'd like you do.' And it was because of this mission statement, as I say, about pushing forward the boundaries of knowledge, they could see that there was something big going on in functional programming, and I was pushing it along, and, they were prepared to, to go with that. I think, if I was applying with that same agenda today, I might get a less… You know, it would be much more about, well, how is this going to fit in with some of our projects? But at the time, that wasn't so much of an issue. So, I regard it as being a, you know, a great blessing, and, and privilege, that Microsoft has been willing to, essentially fund me and, you know, and the work that I do, for 20 years. That's a long time, and essentially a lot of money, to, to fund, you know, this, this, you know, following up this idea of, what… If you really take the idea of purely functional programming seriously, what does it have to give the world? I think it's remarkable that a company has been willing to fund me to do that.

[38:28]
You presumably had to persuade them to, to fund you, did you? Did you have to sort of, do a, you know, some sort of business case or something to, to get them to carry on doing it?

Well, when I was, when I was applying, I said, this is what I do. I didn't have to… I didn't make a business case when I applied. They interviewed me and hired me. Every year we have appraisals. And so, we have to say what we've done, what we're going to do, and our managers look at that, and say, is that what we want you to do? And, by and large that… So, I've never, I haven't had to explicitly make the case for… You know, it's not like… You can imagine, if everyone was sacked every year, and had to reapply for their jobs, [laughs] then you'd have to make a business case. At the same time of course, I mean, I've not been unresponsive to the shift in culture in the lab, you know, so I also do quite a bit of work now on Excel. Because I think… Excel is the world's most widely used functional programming language, but it's such a frustratingly weak language, right? And it's easy to see things that you

could do to make Excel much, much better. Indeed I published papers about that, fifteen, fifteen years ago, about adding user-defined functions, and adding arrays as first-class values. And so I have been, you know, trying to make the case for doing that to a product for a long time. And, you know, I'm still on that trajectory, and you can, some of it has actually happened. This year Microsoft finally released a version of Excel that has so-called dynamic arrays, in which arrays, they have become much more first-class values, and you can write array value formulae without the hilarious use of control-shift-enter [laughs] to enter your formula. Which, if you didn't, you know, if you weren't, didn't already know that's what you had to do, you'd just think was a joke, [laughs] that some engineer invented on a wet Sunday afternoon. So now it's much, much easier. And there's a whole series of stuff backed up behind it that hasn't, you know, it hasn't become a product yet, but I'm hopeful.

Yeah, I mean, is there a strategy for Excel in the future then? Because I mean, it's…

Of course there is, yes.

It's had a long, it's had a long… It's been around for years, hasn't it. Microsoft still expect to come up with innovationa?

It's a hotbed of innovation. Yes. Yes. So, Excel is, you know, one of Microsoft's, you know, longer-standing and most profitable products. And yet, I'm happy to say, that there is a real, you know, a real hotbed of innovation in there. And, you know, and I hope that part of that, you know, part of the innovation will actually land in product, right, actually in the hands of users, will be more of this, user-defined functions and purely functional programming scaled up. I'm hopeful about that.

And the Excel development team, that all happens in, that's all in the States?

That's all in Redmond, yes. So of course there's a lot of to and fro between…

Toing and froing between… Yes,

You know, I've spent quite a bit of time in Redmond. Quite a lot of my colleagues here in the lab are working very closely with people in Redmond. Indeed, you know, if you're using Excel online, you're using code that's written in this building.

[41:19]
And apart from… I mean, apart from Excel then, just talking about functional programming more generally, what are the, what are the key commercial applications of it, and, where, where is it used most extensively, would you say, at the moment?

Well, remember functional programming is, it's a, attack on the entire enterprise of writing programming. So it's not really for a particular class of applications. It's just for, the kind of applications that you would write programs for.

Yes.

Right? So, that's a strength and a weakness. It's a strength because, you know, it has very broad applicability; it's a weakness because, there isn't so much a killer app, right? One thing that it's just stonkingly much better at. But, what you can see is a sustained trend across programming to more functional styles. So, examples might be, things like, if you look at Google's MapReduce, right? That's just functional programming, right? That's what map and reduce… They've taken directly from, those names come directly from the word… Map says, apply every, a function to every element of a list; reduce means, combine the elements of a list together, with a, with a combining function. It's all just functional programming. No side-effects in there.
[42:37]
So, if you look at a language like Java or C#, you will see that they have grown more and more functional features, like, they now have lambdas, they have automatic garbage collection, which was something that was born, and, you know, grew up in the world of functional programming languages. They have… A lot of their type systems were prototyped in the functional sphere, like generics, which is called, in the world of functional programming, is called parametric polymorphism. That was prototyped first in, in ML, you know, a functional language. So… So you're seeing a lot of technology crossover. And more broadly, Pat Helland wrote a paper,

wonderfully entitled, *Immutability Changes Everything*. It's a fantastic title. And, in that he is describing… So immutability, functional programming, go together, right, because, immutability means values do not change. Functional programming you could also call value oriented programming, rather than location oriented programming. Or mutable location oriented programming.

Mm.

So, immutability, functional programming, same kind of idea, right? And his point was, that immutability is showing up all over the place. Think of a log-structured file store. You don't modify the file store in place; you maintain a log of differences to the file store. Similarly with databases, and, transactional memory, you know, you maintain logs of things that have changed.

[43:58]

So that, across, you know, across computer science generally, you can see immutability increasingly showing up. And his point, that's partly because storage has become much cheaper. You know, if you, if you… The way he puts it, accountants never, don't… Accountants don't have erasers. They don't go back to the accounts and change them, rub out the old number, put in the new number. They put in a DIF, because they want to sort of, see the old number, they want to be able to go back to it and say, what were the accounts before? And it's the same, the same with the programming thing. If you've got a, a finite map, rather than modifying it in place, if you want to add a new element, you make a new finite map that shares a lot of data structure with the old one, but both of them are both, are equally available. The old one and the new one, they're both still just first-class values, equally available. So, I think you can see across computer science the trend towards increasing use of immutability.

[44:55]

So, my sort of, my, you know, my sort of soundbite summary is, when the limestone of imperative programming has worn away, the granite of functional programming will be revealed underneath. I think there's a multi-decade trend towards functional programming, and the purely functional language is like Haskell effectively on laboratories in which we are, you know, exploring what the future will look like. You know, I could be wrong, maybe the future won't look like that, and I'm sure it won't

look exactly like that, but it's a, it's a, if you like, it's a sustained experiment in a completely different approach to the whole enterprise.

[45:37]
And, the, the key, as I understand it, you know, the key thing about this immutability is that it means that programs can be written in a much more sort of reliable way, they perform as you would expect them to, easier to write, easier to maintain. Does that mean that they are especially suited as well to applications which need a high degree of security?

Yup.

And, and you know, and ipso facto, applications in finance and, and that sort of thing, where that, where integrity is, is very important.

Yup. Yup, so, I think, you know… Galois, John Launchbury, my colleague at Glasgow, and then the Oregon Graduate Institute where I went on sabbatical, started a company called Galois, which is all about high-assurance systems. So it's not about functional programming per se; it's about high-assurance systems. But Haskell is their secret weapon, functional programming is their secret weapon, that's what they do. So, it's a… So, yes… What makes me think about is this. What limits our ability to build computer systems? Increasingly it's not the capacity of, you know, the speed of the silicon any more, or the amount of memory. It used to be, used to be, how could you fit it into 100 memory locations? Increasingly, it's our own ability to manage the intellectual complexity of huge, you know, 100 million-line systems in which, you know, there's zillions of components all interacting with each other in complex ways. Now, the interacting complex ways bit is the tricky bit, and so, what are the ways that make the functional programs make things easier is, instead of, call it having two procedures that interact with each other, as it were, under the surface through shared mutable state, they can only interact with each other through visible interactions through their parameters and results. So that makes it much more apparent what all the interactions are, and it forces you, forces you, in a way that feels initially painful, but you come to appreciate, it forces you to say, that, the interface of this function has become huge and hairy, right? It's got so many parameters, and so

many results; surely there must be a better way of factoring this abstraction. When that's all under the surface, but it's peeking and poking at these things, that's not nearly so apparent. So, it's a mixture of, you know, the complexity becomes more visible, and therefore, that forces you into, you know, a design, a design process that's more productive.

[48:10]

Also, you mention maintainability. This is… I mean it's a bit… It's… Type systems, static type systems have grown up and become more sophisticated principally in the laboratory of functional programming languages. So… And I think that static type systems are the world's most widely used and effective formal method, essentially lightweight formal verification. But a type is a weak theorem about, a weak theorem statement about what the program does. It takes, if you give it an integer, it will return an integer. That's what Intel array means. So, if you have more powerful type systems that can say more, well then you can verify more advanced properties. And, that means that, as well as giving you additional assurance into what your program does, it also has a huge effect on maintainability. If you take a ten-year-old codebase written in Ruby or Python, you know, written in an untyped language, then you want to make some systemic change to a data structure, and all the people who wrote it have gone, you probably think, I won't do that. With a statically typed language, you're doing that all the time, because you know that, the type system will point you to all the places in the code where that data structure is examined and taken apart. You cannot make… There's a whole class of errors. So, it's had huge, huge effect on maintainability. I think that's the biggest single thing about static type systems in the end. And, somehow, functional languages have been the, the most fertile seedbed or laboratory for advanced type systems in the world.

[49:54]

OK. Thank you for that. So, you know, stepping back from the, you know, from the, from this detail. I mean, what would you say were your key decisions, positive and negative, that you've made during your career that have… And what difference did they make?

Oh, well…

Or was it just happen… A lot of it's…

So, a lot of happenstance. I mean, there's a lot of key decisions, but I wouldn't say that they were clever strategic decisions, right. So there was like, I want to work on functional programming. I did not think, here are ten things, let me pick functional programming. I just thought, I rather like this. [laughs] Leaving, leaving Beale Electronic Systems and becoming an academic, entirely an accident, but hugely important to my life. Becoming an academic at, becoming Professor at Glasgow, hugely impotent to my development as a functional programmer, but it essentially came to me from outside. Coming to work at MSR. MSR happened to be just starting, and has allowed me this incredible freedom. So, I don't want to claim, I wouldn't claim credit for these choices, but these were all choices that did have an important influence. [laughs]

[50:58]
OK. Looking back at, I mean looking back over your career, you've, you've mentioned a number of people that, that you've worked with over the years. Were there any, in particular, any managers or supervisors or colleagues that you found particularly inspirational, and, and what did you learn from them?

Oh yes, I mean lots. So… Right back at, from the beginning, my colleague at University College London, John Washbrook, he, he sort of, essentially took me in hand when I was, first became a lecturer. He was quite an experienced lecturer. Really helpful to me. A very modest and able man. I played a lot of Go with him. Go, a fantastic game, Go. I don't play it much these days, but, had a lot of games of Go with John.

It was very fashionable back in the, back in the Seventies.

Yes.

People playing Go.

Computer scientists like Go because it has these very simple rules that generates this fantastic complexity. And John would regularly… You never lose in an instant in Go; you just, slowly, the snow accretes over your prone body, and you realise that you are now…

It used to be the case that Go was the only game where they, they couldn't sort of computerise it in the way that chess was.

Yup.

Is that still the case?

No, of course it's not. No, AlphaGo nailed that. And AlphaGo was a huge revelation to me. This is when I first thought, OK, so this machine, machine learning stuff, it really has chops, right. Because I really thought that Go would be another ten years out before a computer would beat some of the most advanced Go players. But, AlphaGo beat the best Go player in the world, and that is just an astonishing achievement. So… Anyway, Malcolm Atkinson and Keith van Rijsbergen at Glasgow were very, you know, again, they mentored me when I first arrived to Glasgow, and indeed throughout. My colleague John Hughes, who we're already mentioned, and David Turner, were also very important. And, it's hard to say, numerous, numerous, numerous collaborators since, throughout my, you know… I've co-authored papers with over 140 collaborators, and it's… So it's, it's hard to single any of them out as being, you know, unique, but they have all had a big impact on me.

I think, as you mentioned earlier, I think, the message that comes across is that, for you, collaboration and, getting the energy that comes from sparking ideas off...

Yup.

...off others, is, is very important.

Dialogue, yes, that's it. Yup.

Yes. Yes. Yes.

[53:34]
Do you think you've made any, any poor decisions, or…? You know, in terms of the decisions you've made carer-wise, have there been any… What were the, what were the good ones, what were the bad ones? What would you, what would you advise people, if you had your life again, is there anything you would advise people not to do? [laughs]

Oh it's hard to… That, that's hard to say, right. So that… Even, even things which didn't work out very well, you think, well that was a learning experience. So… I, I really think I've been very fortunate, certainly in my professional life, in fact my life generally. I think that, things have worked out remarkably well for me. After all, look, I might have decided to say, to think functional programming is very cool and interesting, and it might have turned out to be much less impactful than it has, but actually, you know, it has turned out to be right, that. It has become increasingly influential. It might have become, you know, like logic programming, which was also very fashionable at the time and very, you know, look… And, I mean, fashionable in a good way, you know, lots of, you know, lots of very clever people and interesting ideas were floating about, and the, you know, the Japanese had the Fifth Generation, you know, project based on it. But, it's turned into more of a niche somehow. Whereas functional programming seems to have , you know, be enough unspecialised to, to have a broad range of applications. Something I've been, we've been lucky about that. And, you know, that's, that's been great. So it's… It could have worked out differently, that's what I'm saying. [laughs]

You backed the right horse.

Yes. But, more by luck than good judgement. [laughs] Arthur Norman. So thank Arthur Norman, yeah? [laughs]

OK.

And also, you know, it does, it does speak to this thing about, sticking to… I think if there was a piece of advice that I would give to, you know, to other people, particularly other people who have the good fortune to work in universities, pick simple, elegant ideas, and stick to them. Really try to work out where they go. Don't compromise too easily. And if you, if you have that, if you have that luxury, then… Because I think that, making simple ideas into a practical reality, the practical reality turns out to be complicated. Take complicated ideas, and turning them into reality, is often insupportably difficult. [laughs] So, you know, the core idea has to be pretty simple in order to make it practical in reality.

[55:58]
Mm. OK. Over the years you've received a number of awards. In 2004 you were inducted as a Fellow of the Association for Computing Machinery, for your contributions to functional programming languages. In 2011 you, you received membership of the Academia Europaea.

Yes.

What's that? I mean, I think that's one that's probably maybe less, less well known…

Yes.

…I don't know, to, to people?

Yes. Yes, the Academia Europaea is a sort of, group of European academics that arranges sort of cross-disciplinary meetings. I've not really been much involved with them, if I'm completely honest. So, it's a… I… I don't regard being a, a member as a sort of, huge, you know, a huge accolade of me personally. I think it… Respectable, respectable academics across Europe I think can become, sort of senior academics can become members, and I think it does good work. But I, I'm not, I'm not spared enough capacity to play a significant role in it, perhaps I should say, so it's hard to say much about it.

[57:08]

OK. And you're a distinguished Fellow of the British Computer Society. But I think the one that you're most proud of is being made a Fellow of the Royal Society. That was in, was that in 2016, or '17?

I think… I think it was 2016.

Yeah, yeah, that must have been an honour for you.

It was. I was completely… You know, essentially, I never even considered being an FRS for, you know, ever in my professional life until, you know, a couple of years before that. But then… But then after this… There's a sort of significant, there's about, 20, 30, 40 computing FRS's now, you know, depending on exactly how you count. So it is a… And the Royal Society covers science, mathematics and computing, and computing is relatively new to the Royal Society; until fairly recently I think they would have said, we cover science and mathematics, and, you know, computing would have been a sort of, unspoken. But now it's very visibly part of what the Royal Society does. And so I was… And, you know, it's just amazing to be, become a Fellow and sign the, physically the same book that, you know, a few pages earlier, there's Newton's signature. So, I was just completely bowled over, and honoured by, by that. And now, happily, the Royal Society now has a, it has a number of selection committees that choose members, and their commitment to computing has become sufficient, they now have a, since this year, for the very first time, this year, there's a selection committee for computer science. So that's great. Because I think that represents the fact that, computer science is actually a respectable academic discipline, you know, full-throated, and is part, squarely within the Royal Society's remit. And that's quite exciting.

Mm. And apart from the honour, does the, does the Royal Society promote sort of cross-fertilisation of ideas between the, the various disciplines it represents?

Oh lots. Yes. Yes. Yes, so I'm, I'm…

So you get lots of opportunities to do that, do you?

Yes. Yes, I mean, there are interdisciplinary meetings. I've been most involved in education, which we may get to talk to a bit, but, and in particular for example I serve on the Royal Society's Education Committee, which is the Royal Society saying, what about education? This is pre-eighteen education primarily, across all of science, technology and mathematics in our schools in this country. What should we, you know, scientists, be saying about that? And, the Royal Society, being a 400-year-old institution, is an institution that the Government feels able to pay attention to. So that's very helpful. It means the Royal Society has to move very carefully and not just sort of, shoot from the hip, but be thoughtful and evidence-based in what they do. But if they do come out and say something, the Government generally pays attention. So it's a very, it's a sort of high point of leverage. And the Education Committee is the sort of, epicentre of that leverage on education. I mean, the Royal Society also produces a report about climate change, about, you know, science in, you know, legal evidence, and a whole, zillion bunch of other things. It's a great, great institution.

[1:00:11]
And that of course, presumably, ties in to the work you do for Computing at School, for which you are the chair, as I understand it.

Yes.

So those two, those two things are synergistic.

Yes. So Computers at School started a long time ago, that was… Yeah, they are synergistic, but it's, but sort of… You know, CAS started, we started that in 2007/8. So, at that time I was sitting round the lunch table with my children at the weekend saying, 'Well what did you do in school this week?' And, and you know, and particular in computing. And they would speak with contempt about, ICT, which was the subject they were then studying. And I thought, this is really strange. The subject that is so fascinating, that I have devoted my professional life to, and I have spoken about, you know, with, with some passion just now, because I think it's so interesting, is coming over to them as, dull, pedestrian, even worthy of contempt. That was really… So, that, you know… Whereas in biology, say, a professional biologist I think would be able to recognise, in even primary school biology, the subject that they

are professionally engaged in as a researcher. So, this disconnect was worrying to me. And the more people I talked to, the more, more people thought, well, yes, completely nuts, but hey, what can you do? It's the education establishment, you know, nothing will change. So… But since I didn't find anybody who liked the status quo, I began to think, well, if everybody simply got together and said, we should change this, and it turned out there was nobody saying, we should leave it as it is, perhaps, we could change it. So we started Computing at School, which was a kind of guerrilla group of individuals, who just thought the school curriculum in ICT should be, well, reformed in some way. And so CAS started in 2007/8 with four people. It's now, 30,000-odd, and is a, you know, deeply, you know, even institutionally involved in the new computing curriculum, which was launched in September 2014, you know. So we had a, had a, a far larger effect on education at school than we ever thought that we would. And so that has been… It's been like a whole… Again, it's amazing that Microsoft has been willing to allow me to spend time doing that.

So what, so what, what was different about the new curriculum which addressed the issue?

Oh well… So the old curriculum was focused on, well it was information and communications technology. So the focus was on being creative and competent users of, well, ICT. So, it was all about artefacts, things that people had made, and how to use them. Of course that makes it vulnerable to changes in technology. If it's new technology, do you need a new curriculum? Because after all, the technology has changed. And also… And, and curriculums don't change very quickly. So part of the reason my children were contemptuous was that when the curriculum was first introduced, Microsoft Office was, you know, wasn't available in people's homes. So, it might perhaps have been a reasonable thing to learn at school. But, by the stage my children were learning it, they had it on their phones, you know? It's… So… And they were being taught it at school. It does not compute. So, the curriculum was lagging, but also it was focused on the wrong thing. So, the, the big shift of the new computing curriculum is to say, we should think of computing as we do maths and natural science, that is, as a foundational subject discipline, that all children should learn, from primary school onwards, not so much because it immediately and tangibly enables them to, you know, do something, but because it informs their understanding

of the world that surrounds them. So, if you take physics or biology, I mean, you don't need to know anything about combustion or frictional mass or volume to drive a car. You just push the accelerator. You press the light switch, the light goes on. And yet something quite, quite deep in our heart says, everybody should know something about electricity, or combustion, or hydrocarbons, or fuel. You know, just to, may well inform choices about global warming, or about who you want to elect, or… So, if you think about it, it's not all that easy to explain exactly why, but, it's pretty deeply embedded in us. And it's for the same reason I think that we, that all children should learn something about computer science.

[1:04:13]

So the new, it's actually, the shift in the new curriculum is, we're treating computer science as a foundational discipline in the way that we do natural science, and teaching it at an elementary level, as we do for biology and physics, at an elementary level at primary school, and at increasingly sophisticated levels thereafter. And, alongside, being a, a confident and safe and able user of information technologies. So the idea is to equip children with, perhaps especially the ones who will not become the software engineers of the future, but equip children with the, the stuff that they need to deal with successive waves of technology and think, oh it's just a new piece of technology, and I will trust it and, or not trust it, in the same way that I do other technologies, rather than thinking, it's magic, and must work. You see the difference.

[1:05:00]

Yes. So what are primary school children actually learning then?

Well… So, at one level they're probably learning some simple programming. So, just the ability to, once you write some little programs, then, just as when you build a little machine it doesn't do very much, out of Meccano or something, but you get to think, real, more complicated machines work on these same principles. And my machine didn't work very well; maybe real machines don't, you know. So, you know… So, similarly, you get to write little programs, and you think, oh, so that's what is happening in these big programs. It's not magic, it's just more of the same of what I've done. And, so programming is one thing, that's very practical, and can be very tangible, so it all connects with and design technology, you can build things, little robots and things that do stuff, so it's quite engaging. But also, even without

using computers at all, I think, primary school children can learn little bits of, how you might, how… about clever ideas that live inside computer science, like, say, an easy one is binary search. So if I give you a telephone directory, or, in my classroom, I give children telephone directories and I say, 'You look for Jane Smith by searching, page by page, from the beginning, right? You can use any way you want.' What are the 'any way you want' children going do? They're going to open it in the middle and say, 'Oh, that's m. I need to be after that.' Then they'll halve it again, and halve it again. And they'll get there, they'll get to the right page in, you know, log n steps, rather than, n steps.

Mm.

Now that's not hard for a primary school child to understand. It takes a little bit longer before you can think about logs, but actually, you can get some intuitive understanding about how fast these numbers grow, even if you're a, you know, a year six I think. So… But that's an idea that… And, it gives you a sense of, an idea, rather than a piece of technology, or a piece of programming, that is a clever idea that occurs over and over again in computer science, it's a sort of recurring pattern. That's the kind of thing.

Has it worked? How… What are the current generation of schoolchildren saying about it?

Oh, well of course it's very patchy. So, we are… So, we are… The new curriculum launched in September 2014, so we're four years in. But remember that the first child who started with that, won't pop out the end until 2024.

Yes, that's true. Yes. Mm.

Right? So… But nevertheless, in the way of these things, it was all launched in every year simultaneously at once, with teachers who are not trained in the subject, right, they don't have computer science degrees. So there's a tremendous amount that we need to do by way of training and supporting teachers. And indeed, articulating, but since nobody's taught this stuff in a foundational way before, at school level, nobody

really knows. You know, there's no off-the-shelf answer to, what is appropriate for a year four child, or for a year eight child? We had to make those answers up. And there's a tremendous amount of that development going on at the moment. So I think the answer is, it's encouraging, but quite patchy, both because, there are things we don't know, and because, teachers are variously, you know, schools and teachers are variously ill-equipped or better equipped to do this. And, you know, different parts of the country are better equipped or better funded or, funded. So it's… There's a report the Royal Society produced, the same Royal Society, on, called 'After the Reboot'. We produced it in September 2017, October 2017, but it's exactly the state of play, it answers your question in 140 pages of detail. It's a very good read actually. And it led directly, we're saying about the influence of the Royal Society, that report in turn led to the Government allocating really quite a lot of money, about £100 million across the entire nation, to training and supporting existing computing teachers. So it's a good example of closing that, that circle. The Royal Society has had a big influence. So it would initiate this whole computing revolution, it's now playing a very significant part in making a success of it, which I'm very proud of.

Mm. How much of your time do you dedicate to Computing at Schools?

Oh, I don't know. It's a bit, it varies a bit, but probably about a quarter.

Oh, as much as that?

Yes, as much as that.

Brilliant. It's quite sufficient, yeah, yeah.

And it, and it… You know, so, again my, my ultimate boss, the lab director, Chris Bishop, very graciously, at least tolerates my doing this. And indeed I think he's, he's happy to see some of this impact, you know, arising from someone who works for the lab.

[1:09:18]

OK. You mentioned, you mentioned earlier on that your wife is a priest in the Church of England.

Yes.

And, I, I… You yourself contributed to a book, didn't you, in 1999, called Cybernauts Awake!

Yes.

Which was published by the Church of England Board of Social Responsibility, which explores the ethical and spiritual implications of the Internet. So I take from that that you are, you are a practising Christian yourself.

Yes.

And I wondered how that has influenced your work, or the way you go about your work. Is it, is it important?

So I am a practising Christian. We go to church every Sunday. I take my three children to church every Sunday morning, without fail. Dorothy's a, is an assistant priest at the church we go to, and was leading the service last weekend. So, I would say the, the influence that Christianity has on my professional work is kind of indirect. It's kind of to do with… It… You know, it almost has most direct impact when it comes to sort of, interacting with others. So, treating, treating other people with dignity and respect, treating them as if they were working from good intentions. Computing people communicate a lot by email, or bulletin boards. It's awfully easy to… It's a very low bandwidth medium. It's awfully easy to accidentally take offence. So, I think I… I, you know, my self-imposed discipline is to try really hard to assume that I'm the one at fault, when I'm offended or annoyed by somebody's email to me, or, a post that they make, and to try to, you know, contact them individually or build bridges with them. And… And of course, as a, you know… And I think this is true across computer science, I don't know about other subject disciplines, but I think, senior researchers are by and large extremely good at being

respectful and supportive of more junior people or PhD students who will come and talk to them in conferences. They don't sort of brush them off as, you mere worm, but… So I think there's a, there's kind of, generally there's a good culture of, of respect and mutual, you know, mutual honouring of people's professional expertise, good intentions, and commitment, you know. So that is probably the most direct way in which it surfaces. I mean sometimes these things blow up, particularly in online communications, and I try to, try to do what little I can to calm them down.

OK. All right, thanks.

[1:11:50]
Also it's, I think it's… I think, again, all computer scientists would, you know, come from the place, they're trying to make the world a better place. And I think that's, that's not restricted to Christians by any means, right. They're really, genuinely trying to say, what can computers do for the world? And how can I make the world better? Rather than, how can my career be advanced? And I think that's a, that's a good thing generally. The C of E report was a, was an attempt to say, well, OK, so, if you think about the, you know, the world of computers and, from a, from a Christian standpoint, what does, what does Christianity have to say about this? And it's… Since computers have a profound impact on the world that we live in, you've only got to think of the effect of social media and so forth, which rather postdates our report, it's hard to say that it could have no effect, and yet it's quite difficult to talk sense into that space. And I think our, our report was a, a good attempt, but it didn't have much impact. So I think, respect… But, you know, sometimes you catch the wave by, like, some of the stuff about computing education, we caught the wave. We wrote reports, we did stuff, it had impact. There, I think, we didn't catch the wave.

I'm not sure anyone would have foreseen the impact of social media actually. It's, it's just…

No, no, no that's right. But even then, there was lots to, lots to write about, and be, be interested in. But there's… You know, sometimes… When I say catch the wave, I mean, sometimes people say things or do things that have, that catch enough of people's interest that people take action as a result. And you can say, that was

influential in causing things to happen. And that's very random, when that takes place. You know, like… Yes. And there's a lot of luck associated with that. You need to have, you know, some very good intervention, but also one that, that hits a, a particular moment in time. Mm.

[1:13:42]
Well, it kind of brings me on to the final part of our conversation, which is actually about the future. And, just to put you on the spot, how do you think IT will impact society in the next, ten years say?

Oh, I'm very, very cautious about predicting anything, especially when it's in the future. As somebody famously said, did they not. So, computing seems to be rife with unexpected outcomes. Like, I don't think anybody expected mobile phones to take off in the way that they had. I certainly didn't. You know, when they first came out, they cost £2,000 and were the size of bricks. And everybody expecting them to be the, the, you know, the playground of executives, who would have, you know, would be able to afford such a, extremely complicated piece of technology, to, you know, do something as simple as to make a phone call, when you could just pick up a phone, couldn't you. But actually, the power of commoditisation, and mass production, has meant they cost £10. It's just extraordinary. So that was a complete… And, texts, text messaging. 140 characters on a side channel that was initially meant for engineers on mobile phone networks, just to be able to chat with each other so they could say, you know, 'Have you got it plugged into the patchboard in the right way?' has turned out to be a, you know, multi-billion-pound industry. When we already had email. I mean if somebody said, texts, 140 characters, I'd have said, you know, nonsense. Just use email, right? Why would you limit…? So, the world is… Social media, equally unpredictable in my view. WhatsApp and all this stuff. I would never have… So, I have no clue. I have no clue. So, I think the, it's pretty clear that, I think that the whole machine learning thing will have a profound impact. I think we've… I think it's not just flavour of the month. I think that will have a big impact. Because it's, machine learning lets us do things that we can't explain, right. So if I say, how do you recognise a face, right? Before we had all this machine learning stuff, I really couldn't explain to you how I recognise your face. And because I couldn't explain it to you, I couldn't write a program to do it. And

now we have this sort of, alternative approach based on, you know, training, deep neural networks that really is quite successful at those class of applications. It's a bit of a Wild West at the moment, there's a tremendous amount of suck it and see, but I think, nobody's in any doubt that there's going to be some fairly profound impact, but they don't quite know what it is. And, there's plenty to be concerned about, like, like, you know, whether biases in the training data show up in biases in the result, and how you can say why, you know, what… Can you explain to me why you didn't give me insurance? And nobody really can, it's just that the network spat out no.
[1:16:21]
So, I think the, the only, the only… The only prediction I'll make is the one that I started with, which is that I think functional programming will continue on its long-term trend. I think it will continue to be increasingly influential. And I, and I… I'm not quite sure whether it'll, whether in the end we'll all be writing purely functional programs, but I think that, increasingly we'll recognise that that's what we're doing in some guise.

[1:16:51]
And what do you think the biggest challenges and opportunities for the industry will be?

For me, one of the biggest challenges is a terribly mundane one. Why can't we build big IT systems that work, right? So, take the, you know, the NHS's, essentially failed attempt to build an IT system that would connect all our medical data together. Why… You would think, at its root that is not a very difficult problem, right. The fundamental idea that my doctor should have access to records that the hospital has, and vice versa. Not that, how hard can it be? And the answer is, really very hard indeed. And it, you know, it's to do with a zillion things. Very big systems, built by different parties that interact in strange ways. A lot of concern about privacy and who can see what, and delegating responsibility and so forth. But nevertheless, I think, that's a big challenge, that's something we've not cracked. And yet, there's a huge, huge payoff for being able to do that. So, just building, never mind machine learning, right, just building big, mundane, good old basic data, you know, data manipulating IT systems to get the right data to the right place to the right person. That's a challenge I just don't think we've cracked. I'm not an expert in it. I don't even know

what the problems are, let alone what the solutions are. But I dearly wish someone would fix it. [laughs] So for me, that's one of the biggest challenges.

Great. Right.

Oh it's a terribly boring challenge really, isn't it, but… [laughs] Not very romantic, but hey. [laughs]

Someone's got to do it.

It would be good if we did, right? [laughs] It's so ridiculous that we can't.

[1:18:24]
Have any of your children followed you into IT?

So, hilariously, Michael has. So my eldest son, Michael, is, he did a maths and philosophy degree at Oxford, but, he, then he worked for five years for a small company called Semmle, who write software, analysing software. But now he works for IOHK, which is a blockchain company that uses, guess what, Haskell as its primary programming language. And so, Michael is, these days, working directly on GHC. So it's not just, he's followed me into the same, you know, industry, [laughs] as it turns out, probably kicking and screaming, he's ended up working on the same piece of software, which is really scary. [laughs] The very piece of software that I wrote.

[1:19:06]
And what advice would you give someone to, who is entering the IT industry today?

Oh. [pause] I don't know. It seems like, bursting with opportunity, right. So, if you are interested in computing, you are lucky, right. You are interested in and good at something that everybody wants and needs. So, you know, you're in a seller's market. Find something that you really are enthusiastic about. Figure out how to make the world a better place. And go and, go and try and do that. Because you'll probably be able to, right. Whereas if you're working in less marketable, you know,

or less commercially viable things, you may have trouble getting somebody to pay you to do the things you really want to do. But in computing, you can probably find a way to get somebody to pay you to do the thing that your heart really leads you to. And that's a great privilege and blessing.

OK. Simon, it's, it's been fascinating hearing your life story, and, and for you to give us an overview of functional programming. Thank you very much. On behalf of IT, Archives of IT, thank you very much for taking the time with us.

Great, thanks, it's been a lot of fun.

[End of Interview]