LEADERS IN COMPUTING
Changing the Digital World
BCS THE CHARTERED INSTITUTE FOR IT

Our mission as BCS, The Chartered Institute for IT, is to enable the information society. We promote wider social and economic progress through the advancement of information technology science and practice. We bring together industry, academics, practitioners and government to share knowledge, promote new thinking, inform the design of new curricula, shape public policy and inform the public.

Our vision is to be a world-class organisation for IT. Our 70,000 strong membership includes practitioners, businesses, academics and students in the UK and internationally. We deliver a range of professional development tools for practitioners and employees. A leading IT qualification body, we offer a range of widely recognised qualifications.

Further Information
BCS The Chartered Institute for IT,
First Floor, Block D,
North Star House, North Star Avenue,
Swindon, SN2 1FA, United Kingdom.
T +44 (0) 1793 417 424
F +44 (0) 1793 417 444
www.bcs.org/contactus
LEADERS IN COMPUTING
Changing the Digital World
<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Author</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>THE ART OF COMPUTER PROGRAMMING</td>
<td>Donald Knuth</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>THE MIGHTY BOOCH</td>
<td>Grady Booch</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>TALKING TO TORVALDS</td>
<td>Linus Torvalds</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>APPLE TO THE CORE</td>
<td>Steve Wozniak</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>CERF’S UP</td>
<td>Vint Cerf</td>
<td>28</td>
</tr>
<tr>
<td>6</td>
<td>COMPUTING’S TOO IMPORTANT TO BE LEFT TO MEN</td>
<td>Karen Spärck Jones</td>
<td>35</td>
</tr>
<tr>
<td>7</td>
<td>ISN’T IT SEMANTIC?</td>
<td>Sir Tim Berners-Lee</td>
<td>43</td>
</tr>
<tr>
<td>8</td>
<td>SHARING KNOWLEDGE</td>
<td>Jimmy Wales</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>SHAPE THE FUTURE</td>
<td>Dame Stephanie Shirley</td>
<td>59</td>
</tr>
</tbody>
</table>
GETTING TO SPEAK TO WELL-KNOWN FIGURES

Like most journalists I suffer from PIT (pre-interview trepidation) – the people I have the privilege to interview are often at the top of their professions or disciplines, renowned, much-decorated, very very intelligent. Will they disappoint me, turning out to be pompous or overbearing? Or, worse, will they find me out – effortlessly exposing my limitations with one well-turned phrase or insight?

I’m pleased to report that those we’ve interviewed over recent times, many of whom are at the leading edge of computer science are, to a person, thoroughly decent coves. And interesting.

The likes of Linus Torvalds and Sir Tim Berners-Lee rarely speak to the media, so it’s great to get in front of them, if only for a brief time. With others, who perhaps I hadn’t heard of initially, like Grady Booch and Karen Spärck Jones, you find out fascinating things, get background on technologies we now take for granted and even enjoy the occasional laugh (check out Grady’s Quick questions).

Others’ interviews here have been done by my colleagues and they too have fed back to me their fascination with their interviewee and commented on the sheer enthusiasm that genuine experts have. These range from the well-known names – Jimmy Wales, Steve Wozniak – to perhaps less widely known, but just as vital figures.

The combination of the personal and technical insight that a good interview can provide is a heady and revealing one. Enjoy.

Brian Runciman
Publisher (Editorial)
BCS, The Chartered Institute for IT
While he was over in the UK for a book tour and lecture series, Professor Donald Knuth, the author of the hugely respected *The Art of Computer Programming* book series, made time to talk to BCS editor Justin Richards about his life and works.

You’re probably best known for your book series *The Art of Computer Programming*. In 1999, these books were named among the best 12 physical-science monographs of the century by *American Scientist*. How did these books originally come about and how do you feel about the *American Scientist* distinction?

The books came about because, in the 60s, when I began, everyone was starting to rediscover things because there was no one good source of what was known and I had enjoyed writing all the time. I was involved in newspapers at school and magazines and thought of myself as a writer and I realised there was a need for someone to get down all the good ideas that had been published and that we were already forgetting.

This was back in the earliest days, when the number of people actually studying computing was probably less than a thousand. I didn’t see it as affecting the world, but I still thought it was pretty cool stuff and ought to be organised.

Then I thought about who else could write such a book and everyone I thought of I thought they’d probably only mention their own stuff and I was the only guy I knew who hadn’t invented anything himself, so I figured I could be neutral and I could be a spokesman for the other people. And really that was the original motivation.

I started writing the book and, naturally, because I was trying to combine the ideas of many different people, I would see where one person had analysed his method in one way while another, for a competing method, had analysed it another way. So I had to analyse method A according to author B and method B according to author A.

Therefore I ended up creating an original work just by analysing these things and pretty soon I realised there were a whole bunch of interesting scientific approaches here that hadn’t been part of my own education that were really coming together. Over and over again I was really seeing this way of thinking as necessary in order to get the story right.

So, to make a long story short, pretty soon I had my own axe to grind too; I started discovering things too and I couldn’t be an unbiased writer anymore.
However, I still kept to the original idea of trying to summarise everybody's ideas in the fairest, most reasonable way I could.

Now, to be put into that category of one of the best books of the century, that's a little bit embarrassing as they rank me with Einstein and Feynman. I'm not in that league really, I just didn't have as much competition. They had to have a token person in computer science! But still, I worked hard and I think it was necessary to comment on the research so far, but it's a bit like comparing apples to oranges when they chose me to represent computing.

**What is it about computer science that drew you to it?**
I was born to be a computer scientist – I have a way of organising stuff in my head that just seems to make me a good programmer. I think everybody can learn to use computers, but only about 1 person in every 50 is a geek in the same way as I am. That means we can push the envelope and can resonate with the computer. The way we think helps to make it easier for us to know how to construct a machine.

**Why do you think computer science is so important?**
Computer scientists are important because of the way they affect communication and, I'm sorry to say it, also finances. Unfortunately, the world measures what my colleagues and I do by how much our work affects Wall Street. I'm jealous of astronomers, for example, because people respect astronomers for doing astronomy because it's interesting just for its own sake. I studied computer science because it's interesting to study computer science.

The term IT doesn't resonate with me so much – it's the science that interests me. To me the IT is very nice, but it's not something that I'm particularly good at. My wife can figure out what these icons mean and what to click before I can, but there are so many scientific challenges in order to get machines to do complicated, sophisticated things. The ideas are subtle, the questions are fascinating. There are many questions I never thought I'd know the answer to, but gradually we've learned how to solve them. For me I would do it even if there was no money in it.

**So you have a passion for it?**
Yeah, it's like I wake up in the morning thinking I've got to write a program.

**Do you have a muse?**
Yeah, well some days she talks to me more than others. There was a period when I almost thought there was a muse dictating to me.

**In your opinion, what do you think is your greatest achievement in the field of computer sciences?**
I guess the first thing I did well at was when I worked on the theory that goes on behind how compilers work. I worked on the theory that underlies algebraic languages, and, as I was writing *The Art of Computing* book (Chapter 10), I was describing what everyone else had done, but then I realised that there was a way to bring these things together. I didn't know how to explain that in a book, it was too far out, so I published that theory in a paper and other people figured out
what I meant and this became the theory of parsing that’s used in all algebraic compilers now.

But I feel the biggest thing that I developed was the mathematical approach to compare algorithms in order to find out how good a method was. I worked out quantitative ways you could say that one program is going to be, say, 2.3 times better than another one and the mathematics that goes with it and it’s called the analysis of algorithms. It’s what I’m most proud of – in developing an academic subject – but it’s key to the successful use of the machine.

When I came up with this approach, I said to my publishers ‘let’s rename the book and call it *The Analysis of Algorithms*’ and they said ‘we can’t, it will never sell!’ But that’s really what my book is about – it summarises the work of all these people, but it also helps us decide, in a quantitative manner, how good each method is.

**You’ve said on your website, in response to the question ‘why don’t you do email?’ – ‘Email is a wonderful thing for people whose role in life is to be on top of things. But not for me; my role is to be on the bottom of things.’ Can you explain your stance on email and what you meant about being on the bottom of things?**

Someone has not to be tweeting all the time, someone has to be thinking about things which need a long attention span and trying to organise material and build up strong foundations instead of rushing off across the frontier. It takes a long time to put out something that has the right style; I have to really think about it and if I’m going to do it right I have to spend a lot of time focused on it. And I was being treated like an oracle, lots of people from around the world were asking my opinion about whatever, so after 15 years of email I decided that was a lifetime’s worth.

**A previous Turing Lecture speaker, Grady Booch, was very much an advocate of making coding simpler and, according to a blurb regarding your winning the BBVA Foundation Frontiers of Knowledge Award in the Information and Communication Technologies category, you are too. Can you explain why you think code should be kept simple, compact and intuitively understandable?**

I guess you have to go back to Einstein who said ‘keep it as simple as possible, but no simpler’. It’s an illusion to think there’s going to be some sort of ‘royal road’ and everything is going to be easy to understand, but almost never do I find something that can’t be simplified if you have the chance to rethink it again. So every once in a while people have to say ‘well, let’s chuck everything we have and start over again, in view of what we know now’.

There’s a project at Stanford that started a few years ago called the *Clean Slate Project* that said ‘let’s figure out a better way to do the internet’. Things just keep getting added on and accumulate and you realise that there’s plenty of baggage which there’s no reason to have anymore.

It’s like the appendix in the human body, there was probably some purpose for it at one time, but not now. So I think there’s the potential, although I think maybe it’s not possible because the world is so dependent on it, for someone to come along and say ‘let’s start again with the way programs are loaded into machines’. It’s like when Linux came out – that was an attempt at the simplification of operating systems.
Another ideology that you share with Grady Booch is that you have both said that you can appreciate the beauty within coding and programming – what do you mean by that?

I’m thinking of it in several senses of the word ‘art’, but in general the word ‘art’ means something that is done by human beings and is not a part of nature. But then there is fine art, which brings aesthetics into it as well.

In many ways beauty is in the eye of the beholder, but you do something because it’s elegant and hangs together and is a pleasure to read as well as to write or to see someone else’s work; you feel that you’ve got it and you can take pride in it having achieved certain criteria.

Maybe Grady and I don’t agree on the criteria. I mean no two people agree on what’s the best kind of music in the world, but musicians certainly know what they like and what they don’t like and they know when they’ve done something well and that’s the way I look at a program.

I guess it’s down to personal opinion at the end of the day?

Yes, indeed. There’s no algorithm that you feed in and say ‘isn’t this beautiful or what?’ Although people did try – there was a book that came out in the 1930s by one of America’s greatest mathematicians, George [David] Birkhoff, called *Aesthetic Measure* and it was filled with all kinds of formulas and there was a page filled with all kinds of Greek urns and next to each one would be a number which would say how beautiful the urn was.

He classified a whole bunch of different design systems; it’s kind of interesting as number two or three in his list of 100 was the swastika – he was a kind of Nazi sympathiser. I guess it has a greater religious significance in Hindu, if it’s reflected left-to-right. I don’t believe there’s a way to measure it, but he did and some people have tried.

So no one has written a program to work out the beauty of a program yet?

No, not really, although there’s software engineering that tries to do this because they have to measure something – I don’t really know. You know that, as a writer/reporter, you just have to find quantitative numbers to accompany the text – X number of people have died in Cairo, you have to know whether it’s 300 or 315, that’s part of the news story. Qualifying things adds quality. I try to find reason for numbers too, but software engineers are trying to measure how good a programmer is; their bosses know better!

I think numbers are there so people can do a mental comparison and think 20 people have died in that event and 50 in that event so, by contrast the latter event must have been worse. But it’s like comparing apples with oranges, because when you do something to a number then you can start to play games and make the number high even though the thing isn’t right.

You can take education and an educated student and think, well, how are they going to do on this test and out come the books on how to pass this test rather than how to learn science. It’s all about how to get a good score on a science test. And that’s the problem with these numerical things; they don’t always capture
the essence of it. Once you have a way to quantify something then, if your goal is to cheat, you’ll figure out a way to cheat, when the goal really is to learn.

**You’ve said in the past your work is basically about finding a way to sort out the things that are going to last a long time (in computer science) instead of changing rapidly. What do you mean by that?**

Every day I get about one journal in the mail, not including *ITNOW* (laughs), but including *The Computer Journal*. About eight of them arrive in my mailbox every week. So there’s an enormous amount of material out there and it’s good stuff. So how am I going to decide what to put into *The Art of Computer Programming*?

I try to avoid the stuff that’s quickly going to become obsolete and concentrate on the stuff that’s going to have lots of applications. I try to find the facts that aren’t too hard to learn, that are going to be useful for everybody’s toolkit. What should all programmers of the next generation remember? I don’t pretend that I’m right about everything, but I try to sort out the ones that stand out to me, that are unforgettable and that our children should remember.

**So I guess the building blocks of computer science and not so much all the more transient add-ons which tend to follow?**

Yes, but there are still thousands of add-ons that are describable in a couple of paragraphs, and learnable. If something takes 10 pages to describe, then it’s very hard to get it into my book. But if something only takes three pages, is intrinsically useful and I can see how it physically fits in with other stuff, then it’s more likely to go in. For example, we all learned how to add numbers together when we were young. If you think of all the uses to which that skill has been put – it’s incredible. We all use addition every day, in different ways and continue to do so every single day. But still you learned about adding – you learned the concept of adding. There are loads of little concepts like that which go into my book and that’s what I’m looking for. They haven’t all been discovered yet.

Even with adding and computing there’s now ‘adding without carrying or nim-addition’, which is something that was invented in England 100 years ago. It began as a game, which computers can do well, and we could combine this addition with ordinary addition, so one of the things in my new book is to explain to people why we might even want to be teaching fifth graders a new kind of addition because it’s turning out to be quite useful. But it’s not so simple that you can say ‘everything I need to know I learned in kindergarten’ – we keep learning little things that help us take giant steps as we go.

**In 1999, you were invited to give six public lectures at MIT on the general subject of relations between faith and science. Over a decade on, have your views on the relationship between science and spirituality changed and if so how?**

I’m just glad to see that people think there’s more to life than things we can understand and it just seems that, at the time I gave those lectures, it was just coming out of the closet, saying ‘well, computer science is wonderful, but it’s not everything and don’t expect me to give you any answers – let me explain why I think it’s good to still have some things that are mysterious’.

I think there is the tendency as we discover more and more science that we tend to think that now we know everything. But as we think about it more and more
we’re just getting started, I think. The amount that is changing is happening incredibly fast, but still I can see that in 100 years’ time there’s still going to be much more to learn. So there’s plenty of room for humility, but we have still learned an awful lot of stuff we can be proud of.

I had this invitation to MIT and I thought, well, once in my life, if I ever wanted to reflect on this, this was going to be the time and the place to do it. I don’t pretend to be an expert on it; I just didn’t think people were spending enough time thinking about it. I was glad to see how many people responded to it.

Were the lectures well attended?
Well, that’s the thing – it was standing room only! It was a big lecture hall, too. There were six lectures. After the first one it was carried on on Dr Dobb’s telewebcast and it was downloaded an incredible number of times over the next five or six years. So it was definitely meeting some kind of a need. I wasn’t necessarily providing the answers, but I was providing some of the questions that I thought were part of our life – why not discuss these things in public? I was very pleasantly surprised at the numbers who came.

A few years back I gave a talk at Google and again it was standing room only, and again it was about this very subject. And it was a ‘question and answer’ talk like I’m giving for the Turing Lecture. That’s the thing I enjoy, somehow responding to what people ask me more than having a canned thing.

I was going to ask you how you cope with a challenge like that – for a lot of people not knowing what they were going to be asked would be extremely daunting...
(Laughs) It’s not so hard – if I make a mistake, so what? It’s not stressful compared with, say, if you think about Prime Minister’s questions. I think Barack Obama could do something like that, but I don’t think George Bush could have!

In 1975, you published a booklet called Mariages stables, which is a gentle introduction to the analysis of algorithms, using the theory of stable marriages as a vehicle to explain the basic paradigms of that subject. Why did you use that analogy?
It’s a gimmick, but once again it was a series of six lectures that I gave in Montreal. And the theme of those lectures was not faith and science, but looking at the analysis of algorithms and I based it around a mathematical problem I called ‘stable marriages’. You could also think of it as a game between boys and girls where they each have their own preferences and each boy ranks the girls and each girl ranks the boys. And we ask ourselves ‘how can we pair them up according to a number of criteria so that their relationship would be a stable one?’ It’s unstable when there are people who prefer each other rather than their own current partners. There’s always a way to match the boys and girls up in a stable way.

There is good mathematics that can explain why that is true, but also different ways to do it and I can say what method is better than the others and I can introduce the analysis of algorithms into that. And I guess the thing I’d most want to be remembered for (going back to that earlier point) is this methodology of understanding the way these algorithms work. So here was a cute question about matching girls and boys up, which we could solve through concrete mathematical problem solving.
So basically it helps people visualise your ideology in their heads?
Yeah. I could see the average boy is going to have to propose a certain number of times, so it’s not going to take long before we have a statistical probability for an outcome. You can ask a number of questions so the girls get the best deal or the guys do. Good mathematics comes up in the process of trying to deal with this one problem.

It was something I could talk about over the course of six lectures and interact with the audience there. The book (of the lectures) was in French, even though I gave the lectures in English and I don’t know French; they decided to make the transcripts of the lectures in French. So this book came out in French and was translated into English 20 years later. I now know what it’s like now to write a book in a language that I can’t even speak!

You’ve been asked to do the Turing Lecture this year – is that something you’re pleased about?
I’m at a turning point in my life right now where I’m celebrating the conclusion of two big projects, each of which have taken many years to do. So it’s the perfect time for me to be giving this lecture and I told them two years ago that two years from now would be a good time for me – my defences would be down and I’d be retooling ready for the next big project.

I’ve finished Volume 4A of *The Art of Computer Programming*, which I just got my first copy of less than two weeks ago, which is something that I’m proud of and I also finished the eighth volume of my collected papers. All of the papers I’d written over the years were packed into books by topic. For example, one was on typography; I did research beforehand on software for making books beautiful. Another one collects the papers I did on the analysis of algorithms. Still another one has the papers that I wrote for general, non-specialist, audiences about computer science. Plus the last volume of this eight is about fun and games. I was saving it for dessert because these were the papers that I had the most fun writing and that I did purely for the love of it. I love this book on fun and games – I’m not entirely sure why.

On the very same day I finished that book and sent it off to a publisher, *The Art of Computer Programming* book was also sent off to another publisher and then I received my free copies during the same week – it makes me very happy to have them, in both cases. To have completed them, to have gotten through them without getting sick, a world war breaking out or anything like that, is very gratifying. It’s also nice to be able to draw a line under those projects.

Do you think the IT profession has a bad reputation being full of geeks and nerds?
That’s interesting because I was actually proud to be given the name ‘geek of the week’ by a British writer who has a blog called *Geek of the Week* or something like that. I can’t remember his name though. This was about two years ago. They talk about ‘geek chic’ – the word is becoming more acceptable and people aren’t afraid to admit that they’re a geek. Nerd is a little different.

---

My sense of it is that, now that people are identifying themselves as geeks, that word has now brought us to an era where I can say it and people can understand the kind of peculiarities I have. It certainly wouldn't have been that way several years ago.

People who work with words often can’t explain why some words rise and others fall, but this one definitely seems to be in ascendancy right now. I might be wrong about it, but one of the chapters in my book on fun and games is called ‘geek art’ and it just seemed to me to be the right title for it, because it talks about the kinds of art I like to have around in my house.

What, for you, have been the most important developments in the computer science arena over the last five or six years?

To me it’s the fact that thousands and thousands of people are working together. At the end of each year you can ask yourself what the greatest breakthrough discovery of the year was and you can’t come up with anything really, but after five years the whole field has changed. And the reason is, it’s all incremental.

There are occasionally things that are recognised later as being major changes such as the creation of the World Wide Web or something like that, but the year that it happened no one would have recognised it as being so important. Actually it’s more like building the Great Wall of China, with each workman contributing bricks and it’s a teamwork effort where so many people are involved. It’s all about pushing the envelope and learning from one another, so I think that’s the way I perceive it.

Looking forwards, what do you perceive are the biggest issues/challenges computer science faces, now and in the near future?

The challenge of how we can go to sleep at night when there are so many things left to be done! As an American I’m a big admirer of (and also slightly jealous of) the British health care system, from what I know of it (my grandson was born here), but I think so much more could be done by having better health care records and better ways of describing and visualising combinations of symptoms and combining statistical methods as well as visual methods. This would help doctors to understand things more clearly and quickly.

Not to mention the ways in which computers can help biologists to design better drugs. Everywhere you look there’s something to be done which needs a good programmer to help achieve it. There’s no shortage of challenges and little chance of ever running out of them.

I started off by saying that 1 person in 50 is a geek like me, but I think there might be a geek shortage in years to come. I might be wrong – the next generation could be 10 per cent geeks, but I doubt it. In order to do these things that computers need to do, we’re going to need the people to program and run them who have these weird talents.

Do you think the old artificial intelligence chestnut will ever be solved?

I don’t think we’re anywhere near this singularity, but there will gradually be a coming together of men and machines. I must say that my colleague,
John Hennessy, the president at Stanford [University], has said he thinks there’s going to be a computer meltdown in five years, like the financial meltdown, because we’ve become so reliant on them; people are relying too much on them. There’s definitely going to be a time where people have forgotten how to do things, all because of relying on machines. We’ll forget how to do stuff without machines and that’s going to lead to some crashes.

You once said ‘Everything about computers today surprises me; there wasn’t a single thing that I could have predicted 30 years ago.’ If you were to set yourself up as a sci-fi author, what predictions do you think you’d make about life in 50 years’ time?

I’m glad you found that quote. Anyways, it’s true!

Pessimistically, I don’t see how we’re going to solve the energy situation unless there’s something like breeder reactors that dispose of waste properly. There’s something called the Jevons paradox dating back to England in the 19th century. [William] Stanley Jevons, I think his name was. Someone worked out how to run the railroads 10 times more efficiently than they had done in the past and as a result they used 100 times more coal because everyone then started to use the railroads to transport things. In other words, once you made something more efficient, then people used it much more. You don’t just say we need X amount of oil to do what we want. What happens is that if we had more oil, we’d be doing more with it now. And what happens is our appetite is never satisfied. Hence I don’t see how our energy requirements will ever be met.

The optimistic scenario is that everyone enjoys doing analysis of algorithms and enjoys doing beautiful computer programs – wouldn’t that be a great future!

Links
Donald Knuth’s Homepage www-cs-faculty.stanford.edu/~uno/
Turing Lecture 2011 www.bcs.org/turing/2011
Perhaps you could tell us what you are working on currently?
There are two areas that I'm giving my attention to. The overarching thing that brings them together is addressing what one can do to improve the efficiency of developing, delivering and evolving complex software systems.

There is a gap between vision and execution; we can dream up many things that have a software element to them, but to turn that into running systems is a challenge. Pure computer science can limit us, so can social, legal and moral things. But where most organisations stumble is at the cusp of design and organisation – those are the two places where I'm spending my attention.

In the area of design, one of the greatest advances in the last 10 years is the observation of design patterns, the ability to look at things at a higher level of abstraction that transcends the original programming language. Not that languages are unimportant, but to address complexity we have to move up levels of abstraction.

The Hillside Group, of which I'm a founding member, has begun to catalogue several thousand design patterns, so that’s a maturing discipline. We know to express patterns we needed to catalogue in the physical networking domain and the abstract logical domain. Where we lack is in our ability to describe significant or architectural patterns.

In civil engineering, there is a history of architects learning by studying the works of their masters – Frank Lloyd-Wright, Christopher Wren, Frank Gehry. There are books that catalogue the different genres of buildings and observe the common kinds of architectures. No such reference exists for software engineering.

In software, people tend to be brought up in one domain with almost no knowledge of others. Nor have they studied the masters. I've found that only one or two institutions in the computer science world have a reading course. This is much more common in other disciplines.

So my goal is threefold: first, to build a handbook of software architectures with the idea of codifying the architecture of a number of interesting systems – historical,
economic or technical interest. Then to extract the patterns we find, because every system has an architecture, even if just accidentally, and the systems we have today have their architectures for historical and technical reasons. The third goal is to provide a reference to move us up a level of abstraction so that people who wish to become architects can have a reference.

That’s one area. A lot of what I do for IBM is to mentor, to work with organisations to define and transform their architecture and thus their organisation along the way. The other area is the problems of organisations. Most economically viable software is not written by an individual or even a small team, but by many code warriors working with other stakeholders who are often geographically and temporally dispersed.

The geographical is obvious, but the temporal distribution is under-served and little understood. I may build something, and there are people that follow, maybe even a generation later, that may want to modify or adapt what I have built.

We know that code is the ultimate truth, but not all the truth. There is entropy, a loss of information, from vision to construction, so even though I may stare at some code I don’t have access to the rationale or the patterns that sweep across the individual lines of code.

So we have information loss. So I have tried to define what we can do to improve the developer experience using a thing that Alan Brown and I have called collaborative developing environments. It’s not a killer app, but a weaving together of a hundred small things – the things we know from social networking sites like Facebook, Slashdot and Sourceforge. So we want to produce technology to help us deal with the geographical and temporal distribution.

**What do you think we get wrong in software development at the moment?**

It’s the design and organisation struggles I’ve mentioned. If I have built something a hundred times, it’s relatively easy to build it the hundred and first time – but most organisations are building novel things and we don’t have good models for how we should build them.

There have been few studies of what hyper-productive teams really look like. Most organisations tend to build systems that model their organisations and their organisations tend to model their systems over time. Therefore, when someone wishes to innovate, there is resistance to change because this has been ossified by the form of the organisation itself.

I wish I’d been trained as a sociologist or a psychologist because many of the problems I encounter are not technical problems; they are social ones with a deep technical element.

**Usability is always an issue. Will we get the raising of abstraction to a level where people can intuitively use software?**

If I took somebody from the 1800s and put them in this room now, they would have incredible cultural difficulties because they wouldn’t understand the things around us – your iPod, the projector and the like. But we are now
LEADERS IN COMPUTING

birthing a generation of people who never knew a time when the internet didn’t exist.

So while someone in the 1800s may know about the care and feeding of horses, for example, so it is true with someone born in this generation who grows up surrounded by computers. There is a raising of the tide so that for the next generation texting, browsing the web and booting a computer is like turning on a coffee machine – feeding a horse, as it were.

So in every generation this culturisation happens. It’s part of a given environment. Is it the right one? Are the interfaces we see on the web and the like the perfect ones? There is no such perfect thing in engineering discipline, but what we have is a result of historical, hysterical and emotional confluence. What it is, is what we have.

Software becomes part of the atmosphere and the usability we struggle with today will just become second nature.

**BCS is pursuing professionalism in IT. What are your thoughts on this?**

My life was saved by software. When my nephew died at age 20 I had a CT scan, which revealed I had an aneurysm. If I had been born a generation before, there would have been no such diagnostic tools. So I relied on the skill of the medical staff around me and the technology. The CT scan and the software inside it saved my life. I probably wouldn’t have felt comfortable if that had been developed by a hoard of people who are just script-kiddies for which there was no intentionality in building that system. I’d probably be dead.

I’m a frequent flyer too. Most of the planes in which I fly and the air traffic control systems I depend on would cease to exist without software. So on a daily basis I put my life in the hands of people who have written software. We are in an age of increasing, if not total, dependency on software. As that increases it calls on us to show increasing professionalism in our space. It would be unthinkable here in London to have regular reports on buildings falling down – and yet that’s what happens in the software world. There are economic pressures that push organisations towards building software that is better, faster and cheaper. But we can do better.

In a way it’s like what we saw at the beginning of the railway industry and utilities, in which there were lots of people pursuing it because of its potential for making money but not a lot of professionalism in the process. As an industry we’ve struggled to define what it means to be a software developer. There are a lot of amateur developers, and I don’t mean that in a derogatory sense, but there is a lot of software produced by people that haven’t been trained. We owe it to the community to capture best practice. This will increase the level of trust the world has in our products. So professionalism is key.

It is a tremendous privilege and responsibility to be a software engineer. It’s a privilege because the things we do change the world; it’s a responsibility for the same reason. The world relies upon us.
What about certification?
I’m mixed on the area of certification because we don’t have a good grasp on the body of knowledge we wish to certify. There was an attempt at this with the software engineering body of knowledge by the ACM and others, but I left it disappointed because we couldn’t even agree on what simple terms meant. The International Association of Software Architects, of which I’m a board member, is attempting to find this body of knowledge.

BCS is celebrating its 50th anniversary this year. What development in computing do you think was the most exciting or groundbreaking in the last 50 years?
I can’t pick just one. The public may identify one thing, such as the web, but as an insider I think the progression of our field has been the progression of science. More evolutionary than revolutionary. People may point to the development of the web, but that is not a point in time. I had my first email address in 1979 via the Arpanet and Arpa published a little book, perhaps 15 pages, which gave everyone’s email address in the world. We couldn’t quite do that these days.

The web has been evolutionary. Similarly, let’s go back in time. I was at Bletchley – first can I say thank you to the UK for inventing the computer? – and if you look at Colossus we can look at the things going on around it and it’s still evolutionary, not revolutionary.

Design patterns are the most important for me in the last 5 to 10 years. The development of software has parallels to the development and maturation on other sciences.

Which past discovery would you have liked to have made yourself?
Fire.

We have a problem attracting students into computer science – I understand it’s the same in the US. What can we do to try to reverse that trend?
Earlier this year I gave the keynote at the ACM Conference for Science Educators and that very question was asked. I have tracked what the ACM, IEEE and others have done in terms of recommended curriculums for universities and it’s clear that the problem starts very young. We in some ways alienate kids from interesting stuff in software and warp their minds.

Software in schools usually means knowing how to surf the web, get a broken PC up and working or install the latest patch on Windows. This is not computing. We are missing teaching the notions of awe and beauty, especially with regards to women in computing. Let’s teach them how to program a single person shooter game – well that’s not exactly thrilling to half of our species. So we have to start early to teach, not just skills, but the joy, beauty and awe of software. I love what I do – I don’t need to do it, but I love it because of the joy and awe and beauty.

Vint Cerf thinks we don’t celebrate the role models in computing enough.
This goes back to the handbook of software architecture. Along the way I’m interviewing the architects, because there’s a tremendous human story behind all of this. In my life I could count maybe a half dozen people that I consider world class architects – people at the Gehry, Lloyd-Wright, Wren level, and yet most of the
world, or at least most of the people in our space, don’t even know them, let alone celebrate them.

Microsoft had a self-serving book a few years ago called *Programmers at Work* in which they interviewed a variety of programmers who were mostly building Microsoft products. It was fascinating. There’s a similar book out now called *Beautiful Code* from O’Reilly, which begins to celebrate the lives of individuals.

There’s this horrible stereotype of these geeky, smelly, poorly dressed people that sit in dark rooms in front of sticky keyboards. Maybe we need a TV reality show about developers and how wonderful they can be. Pitch it to the BBC.

**I interviewed Linus Torvalds recently and we got onto the subject of Ray Kurzweil’s singularity idea. He thought it was over-hyped. What do you think?**

I agree with Linus. In a recent lecture I projected ahead to 2031, coincidentally the 50th anniversary of the founding of Rational. In it I said that we as an industry are the antithesis of high energy physicists.

Richard Feynman did incredible lectures about what they do in their community, which is to tease out and make visible the rules and laws at a tiny level. In software we have intense complexity and are trying to squeeze it into a space so that it is invisible. The opposite.

So as I look ahead there will be increasing dependency on software but we want to make it disappear. Development isn’t getting any easier. But looking at Ray’s issue, I don’t know if consciousness is an accident of biology, but can I achieve the illusion of intelligence? Probably. Less so in where Ray is headed, but more so in Marvin Minsky and Rodney Brook’s direction, that one can achieve amazingly complex behaviour from a few simple things.

Will that cross the threshold into that singularity? Maybe, but it will probably take longer than what Ray has in mind. It’s going to be subtly different. I have more computing power around me now than existed in the world about two decades ago. I have half a terabyte of storage either on my body or beside here on my laptop. It’s not going to go away – that will increase over time. I expect there will be a point in time, evolutionary again, where they become more tied to me.

Perhaps I’ll carry glasses around with me that will record what’s going on, maybe with facial recognition so that when I see you again I’ll know who you are. These things would become an extension of my consciousness. I just won’t see the boundary between me and these devices anymore. Will we have computers that by 2031 will pass the Turing test? My guess is probably not, but we’ll be pretty close. We’ll be able to build programs that will be able to simulate stupid presidents.

**Brian Eno gave a speech on generative music a few years ago. He liked the complex things arising from simple rules too, interestingly. Do you think we use software well in the creative arts?**

It’s created opportunities for new mediums of expression. My wife is a fabric artist and she increasingly uses her Mac and Photoshop to create her designs.
We are seeing a generation of folks who are discovering the computer as a creative medium. In Austin, Texas, there is a museum of digital art.

The software inside synthesisers has created the ability for untalented performers, such as Britney Spears, to produce vapid music more rapidly. Good side, bad side. But it’s also allowed us to produce music that’s quite interesting and striking.

Creative types will navigate to whatever medium allows them to do creative things.

I contacted Frank Gehry’s company, who do some incredible buildings, and asked if they copyright their designs. They said no. They trademark them, but don’t copyright because they make their money because they can build these buildings when no one else can. Gehry has a software company and they sell software to their competitors. His buildings could not be built without the software. They have 3-D modelling programs and programs for doing wind, earthquake and water analysis.

Who in the IT industry, or outside it, inspired you or was a role model for you?

There are people in the industry who inspired me. When I was 10 or 11 there was an article in *Life* magazine about a robot called Shakey, and in it they highlighted the work of Marvin Minsky. I thought this was so cool. I knew at that point in time that I wanted to go into computers.

So I scoured the literature, of which there was not much in 1966, and I read several books on digital electronics. There is a British computer scientist called George Walter who built these little robots, so I was influenced by his work.

Age 12, I built my first computer, scrounging parts here and there. And here’s the postscript: I convinced the Computer Museum that they should include software to preserve source code for future generations. So I relayed this story to the board of the museum and the curator said, ‘turn around and look in the box behind you’. It was the original Shakey that had inspired my work.

Dr Minsky, Dr Brookes, Dr Hoare, Edgar Dijkstra – these are my heroes.

What recent developments by others have impressed you most?

I am continuously delighted by Apple and their ability to build wonderful products. At home I have banned PCs because I prefer to use operating systems that work. I have only Linux or Macs in my home network. I have a 2TB file server; two T1s that come into the house, devices that allow me to look at my security cameras from a distance... so I’m very, very wired.

If not in IT, what would you be doing?

I would have been an itinerant musician or a priest. I play a variety of musical instruments. My first instrument was an accordion, which, at 15, I learned are not babe magnets, so I switched to guitar. I built a synthesiser and got into that until the mid 90s when I realised I was tired of booting my instruments to make music. So I abandoned all of those and went acoustic. I’ve gone to the Celtic harp, the hammer dulcimer, the flute, the harp and I sing.
What books have you read recently and, without wishing to encourage stereotypes, who are your favourite science fiction authors?


In terms of science fiction, I love the classics; Asimov and the like, and Terry Pratchett. I'm a Monty Python fan too. I also read physics books and maths books for fun.

In films, I'm a James Bond fan, I also love a good chick flick and a good cry. I was delighted last time I was here because Drew Barrymore and I shared an elevator.

Just going off at a tangent briefly – Asimov posited the science of psychohistory, predicting the behaviour of mass populations statistically. Any chance?

There was a play called a *Search for Signs of Intelligent Life in the Universe* with Lily Tomlin, where she has this line ‘reality is just a collective hunch’. So, insofar as we can identify what that collective hunch is, then we're well on our way.

Quick questions

Mac or PC?

Let me think... I was an Apple 2 user and I have never willingly bought a PC.

Are you a geek or a nerd?

I don't the know difference between the two. I'm a renaissance man because I have a lot of interests. I think I was born in the wrong century. I think I would have flourished in the time of the Renaissance – of course I'd be dead by now, but I would have had a smashing time.

Smartphone, PDA or iPhone?

I carry a Samsung phone because I travel so much internationally and I need a quad band phone. I would love an iPhone, but it doesn't serve my needs for travel. I carry a Palm as well, and I used to carry a device that fused these together, but the technology changed at different rates so I decided to get a good separation between the two devices.

How would you like to be remembered?

Apart from for the invention of fire? As a kind and gentle man – that’s enough.

One piece of careers advice?

I've mentored six people in IBM, so I do this a lot. What I say is follow your passion, because if you don't you won't have any fun, and if you're not having fun you should be doing something else.

Links


Brian Eno’s generative music www.inmotionmagazine.com/eno1.html

Ray Kurzweil’s singularity idea www.youtube.com/watch?v=cc5gIj3jz44

Computer History Museum www.computerhistory.org