An Interview with

CHARLES ANTONY RICHARD HOARE

OH 357

Conducted by Philip Frana

on

17 July 2002

Cambridge, England, U.K.

Charles Antony Richard Hoare Interview

17 July 2002

Oral History 357

Abstract

Sir Antony Hoare is Senior Researcher at Microsoft Research in Cambridge, England, and Research/Professor Emeritus at the University of Oxford. Hoare is the recipient of the A.M. Turing Award for fundamental contributions to the definition and design of programming languages. He has also been awarded the Kyoto Prize in Advanced Technology for pioneering and fundamental contributions to software science. In this oral history Hoare recounts his personal involvement in the development of academic computing science and education at The Queen's University, Belfast, Northern Ireland, and at the University of Oxford. He discusses his long-time interest in building bridges between university computing science departments and industry. Hoare also details his current work at Microsoft Research in applying assertions and other scientific techniques and theory to industrial operations. He discusses his advocacy of assertions in the maintenance and transformation of legacy code. Hoare also comments on a number of other subjects, including machine translation of languages, artificial intelligence, reasoning under uncertainty, software design and reliability, and project management. The interview includes a discussion of the problem of the preservation and interpretation of code.

This is an oral history with Tony Hoare taken at Microsoft Research in Cambridge, England, on July 17, 2002. This oral history is conducted for the CBI Software History Project.

TAPE 1 (Side A)

Frana: How did you come to be here at Microsoft?

Hoare: Partly by romantic concerns I started in industry. I worked for eight years for a small British computer manufacturer, worked for over thirty years in the University, and I thought it would be nice to create a sandwich by returning to industry for the final years of my working life. In particular, the British universities operate a strict retiring age, which is 65. I didn't feel all together shot in my bolt by then, so an opportunity to continue working in computer research was an opportunity that I couldn't really resist, even though it meant moving from Oxford to Cambridge, which is quite a significant move, on a personal level. I think an offer from Microsoft was particularly relevant, because this is a company that has software as its primary product, and I felt that if there was any prospect of some of the research ideas that I'd been working on bearing fruit, and other long-term research ideas emanating in general from the academic community, Microsoft was the place to be. I heard a talk from Nathan Myhrvold who was the founder of the Microsoft Research division, and his description of the academic freedom and the long-term nature of the research really attracted me, because I felt that universities were coming under increasing pressure to concentrate on short-term research and research with short-term industrial goals. Microsoft was not taking the short-term view and I felt I would like to help contribute to whatever view they do take.

Frana: Could you tell me a little bit about the grand challenges that you described as we were walking back from lunch, and how you came to see that as a valuable exercise here at Microsoft.

Hoare: Well, computing science research has been stimulated by a fairly large allocation of funds to the subjects of Information Technology and Communications, but the funds have usually been earmarked, and come with certain political strings. The research has to conform to the currently politically fashionable goals of being fairly close to commercial application, having a good prospect of giving short-term economic benefit, and having a clear path to application. I felt, and indeed a number of people feel, that science doesn’t progress only by a series of short-term goals and that, although they can contribute greatly to the progress of science, one needs also to consider long-term goals of a kind that transcend individual research projects, while working with limited budgets on fixed time scales. In most mature scientific disciplines, the long-term goals are generated as it were from the logic, the structure of the science itself. They’re motivated by curiosity and enthusiasm, and they are somewhat detached from the eventual application which the knowledge will prove to be relevant for. Computing science in general has not had the privilege of determining its own goals in this way, and I thought it would be nice, it would be a challenge shall we say, to try to complement existing research motivations with a few well-chosen grand challenges.

Frana: So, do I have it right then that short-term goals produce evolutionary growth, but to produce revolutionary growth you need longer-term goals?

Hoare: I think that’s right. In selecting longer-term goals, one should create opportunities which the short-term goals would not allow. The short-term goal always has to consider, ‘How do we get there from here?’ It is always based on the economic situation as it is now, and it plots an evolutionary path that in some cases is heavily constrained by legacy—in the case of software, legacy software; in the case of communications, legacy hardware. It is constrained in practically all cases by the legacy beliefs and attitudes of the users of computers. So a grand challenge that pursues a revolutionary goal, that will change the way we all think of the subject, is to be preferred over one that is going to be achieved anyway by evolutionary steps.

Frana: So your experience at Oxford was of less and less long-term freedom?

Hoare: I think the freedom at Microsoft comes from the fact that we don’t have to create a stream of grant proposals. Still, I think at Oxford I was really very fortunate. I did very much welcome the opportunity and the encouragement to work with Industry in trying out some of our ideas. I had some very successful collaborations with IBM, and with a

British start-up micro-processing company called Inmos. They really inspired a lot of the work that now, even from a theoretical point of view, I'm most happy with.

Frana: Did you have connections with Microsoft when you were at Oxford?

Hoare: No connections with Microsoft at all.

Frana: Other than their use of assertions?

Hoare: That was something I discovered later. These things are rarely attributed to their source when they come into use in practice.

Frana: So they had no idea that assertions were your specialty?

Hoare: Well, I think quite a number of them did, in fact. When I first arrived, I sent around a general missive asking people about how they were currently using assertions, and I found they were using them in a lot of different ways. They were using them chiefly for program testing, using the assertion as a test oracle the way that engineers use instrumentation on a test bench to detect the occurrence of errors as close as possible to the place at which they occur. That rather surprised me, because during the academic phase of my career I was very antagonistic towards testing. I was concentrating on the potential role of assertions as a means of verifying computer programs, avoiding errors altogether, rather than merely trying to remove them by debugging. I see that the use of assertions for testing, the use of assertions for specification, the use of assertions for proof, and the use of assertions for optimization of code and for its documentation—all of these are contributing to each other. If we can expand the culture of using assertions, maybe we could get the same assertions used for more than one of these objectives, and so exploit the effort that goes into constructing and formalizing them more effectively. That's a message I've been preaching inside Microsoft for some years now.

Frana: So these assertions are then left out of the final compiled code that finds its way to the user?

Hoare: The original view was that. After the assertions have served their purpose during testing, like engineering probes, they are removed before delivery to the customer. Actually, computers delivered to the customers are now so fast and so large that it's perfectly feasible to leave a lot of the assertions in the code to run on the customer site. They also protect the code against the possible occurrence of errors that might otherwise derail it.

Frana: Now this is a related prong, but I read a piece you wrote entitled 'Legacy.' It is about protecting legacy code with assertions.

Hoare: Yes. I've come to learn to live with, and even to love, legacy now. A major part of the work of a Microsoft developer is introducing changes and additions to the functionality of legacy code. At the same time they have to read it and understand it.

When they have understood it, they are inclined occasionally to improve it; improve it from the point of view of making it more comprehensible, improving the documentation, adding more assertions, and so on. It is the maintenance of legacy code that is the strongest motivator for adding additional redundant assertions to the code in the first place. So even legacy code can be used as an argument for the more effective use of assertions.

Frana: Thank you for bringing me a bit up to date on your current research here at Microsoft.

[pause]

Frana: One of the things the sociologists tell us is that expertise requires a credible body of knowledge. When you were starting out did you have the sort of difficulty that your friend Dijkstra had, for instance, even in calling yourself a ‘programmer’? He once said that he couldn’t get a marriage certificate at first because the authorities didn’t recognize the word ‘programmer.’ He had to identify his profession as ‘theoretical physicist’ instead.

Hoare: No, I never had difficulties with the word ‘programmer.’ Initially, the word programmer would have meant somebody who constructs programs for television shows, but once that confusion was got over I think it was accepted in my passport without difficulty. My wife is a programmer too, so I expect our marriage certificate has the word programmer on it twice. When I first moved into industry, programming wasn't regarded as a profession, or anything to which a pre-existing body of knowledge could be applied, because there simply wasn't any pre-existing body of knowledge. We just read the manual, perhaps attended a short course, and started writing code. We thought of it as something anybody could do. You do require talents—being able to imagine the execution of the program, which has not yet been written or executed, and planning how the various parts of the execution are going to fit into each other. But for those who could do it and liked doing it—and we usually did—it wasn't considered really any great challenge. It was really only when moving from industry into the university in 1968 that I began to think of what you would teach. How can one turn this subject, which every practitioner had taught himself, into something that you could teach to others? Actually, I read an article by Peter Naur where he summarized his view of what computing science education should consist of, and that influenced me greatly. Then, when I actually had to set up a graduate degree and then an undergraduate degree in the subject, I was able to develop my ideas still further.

Frana: Tell me then a little bit about Belfast. Was computing science taught in the mathematics department at that point?

Hoare: The computer science department at The Queen's University in Belfast in 1968 employed four people, and it taught a Master's course. It was a member of a group of departments in the School of Applied Mathematics and Physics.

Frana: And how quickly did it develop into a separate branch with carefully defined divisions? Did you have a computing science that was highly mathematical and another in a school of business and economics?

Hoare: I think that in Belfast, as in many universities in Britain, the computing science department was fairly independent of engineering departments. They didn't tend to be dominated by electronics. Nearly all of them were completely independent of business and sociology departments. The most likely association was a mathematical grouping. But they could be created by an act of the vice-chancellor more or less out of the blue, and with freedom to develop the subject with whatever academic freedom people have to develop their own subject, in the same way that other scientific subjects define themselves.

Frana: Did any of them emerge in the way that Carnegie Mellon's did? Did any of the departments emerge from business administration or operations research programs?

Hoare: I can't answer that question with any authority, but I believe at the London School of Economics the computing department had that sort of origin.

Frana: Was there great controversy over the 'intellectual respectability' of computer science when you were at Queen's University?

Hoare: I think we definitely felt that we had to justify ourselves. There's a terrible academic pecking order in the universities which is sort of pervasive: physicists don't have to justify themselves to anybody, biologists are pretty secure too, and mathematicians. Computing—yes, that's one of the things we have to do, to gain the respect of our colleagues. But then there are subjects that even we can look down on.

Frana: I know historians spend an awful lot of time discoursing on their respectability, but then we have the political scientists to pick on.

Hoare: Yes.

Frana: Were the early curriculum guides of the ACM and BCS useful to you?

Hoare: I did look at ACM curriculum guides, and I think there was something similar from the BCS and they probably had a good influence initially. I think that once the university has set up a department, it also tends to be dominated by legacy, and only make small changes.

Frana: If you were to point out the greatest novelties of the department that you set up at The Queen's University?

Hoare: At The Queen's University I don't know that I thought we were being very novel at all. The courses developed gradually because we thought of them together as a joint honors school with mathematics or physics. That gave us the ability to start small. In a traditional university, a given subject could only offer one out of three courses in the first year anyway, and then a couple of courses in the second year; then a couple courses in the third year, and then you already have a joint honors degree. The teaching grew as the staff grew, following the interests and experience of the staff.

Frana: When did it get a doctoral degree program?

Hoare: A lot of universities in Britain started with a doctoral degree, and Belfast had doctoral students in computing science even before I arrived.

Frana: What about accreditation? Over your tenure at Belfast were there accrediting organizations?

Hoare: The universities in Britain did not have accrediting organizations. They had independent governing bodies. Later on, of course, the professional societies would organize accreditation visits, but I didn't participate in those until I'd been in office for some time.

Frana: To your knowledge, were models adopted from other professions? I know that you have often compared computer science to engineering.

Hoare: Yes. I have taken engineering as a sort of ideal toward which computing science aspires, because engineering has a clear intuitive, common sense content. It also has a scientific content with a mathematical basis. One of the charges I've always incurred, justifiably, is of concentrating more on the mathematics than many people would find welcome. I think it's a bit like programming. There is no textbook on how to set up a course on a new subject at a university. There is no textbook on how to be a professor at a university, so everyone has the freedom to work out their own salvation, which I've enjoyed doing.

Frana: What about this idea of a programming 'priesthood'? How old an idea is that?

Hoare: Well, I drew that analogy between programmers and the high priesthood in an article which I wrote about the subject. The numbers of points of similarity certainly are pretty arcane. The instructions that the high priest gives to the person that consults him are usually in the form of some mumbo-jumbo—slash-slash dot-dot, something or other—and I contrasted this with the message that I wanted to get across: that programming was a profession and should develop as a profession, and that we should copy some of the practices of professionals in other disciplines. Keeping up with the literature in the subject, for example, is one of the duties that the responsible professional enjoys. Programmers on the whole don't. They are too busy to read the literature.

Frana: You have said that specialization is something of a precondition to a full-fledged profession. How then do you create mechanisms that nurture general-purpose minds, people who are able to intuitively leap across the boundaries of these specialties? Knuth talks about becoming proficient in two divergent sub-sub-specialties.

Hoare: I think that's the best we can do, yes. The courses which I planned in Belfast and later in Oxford, include the major programming paradigms: functional programming, procedural programming. They also include hardware design, because that requires the same kind of intuitive knack of inventing algorithms and techniques for achieving a goal that apply in all those areas. So one hopes that the student gradually gets the knack.

Frana: Knuth argues that maybe two percent of the population is really suitable for professional programming. It seems to me from the pieces that you've written that you don't agree. We can all be trained to a certain degree.

Hoare: Yes. I would be condemning thirty years of my professional academic life if I didn't believe that some talents could be developed. It's extremely difficult to measure, extremely difficult to make a convincing case that a little bit of training here in different subjects can add up to a well rounded professional. I am surprised and very gratified by the fact that very few people ever regret whatever education they've been given. The same is true with computing. The graduates will say, quite honestly, that many of the things that they learned they have never been able to apply directly, but I don't think any of them ever regret learning it.

Frana: Well, these things can become helpful many, many years down the road without you ever being conscious that it’s something you learned so many years ago.

Hoare: That’s right. I think when you’re searching for a solution to an intractable problem, you’ve got to have a very broad search space and very good criteria for cutting down the search space. An education is certainly good at giving you criteria for cutting down the search space, but you don’t want at the same time to suppress the imagination, which gives you the search space to cut down. It occasionally comes up with a miraculous solution.

Frana: Those do seem to be competing motivations: structuring minds and yet helping them be flexible in ways that allow them to be built upon. In your last note to me you said, ‘I meant to say that research was a craft, not that programming is.’ I assume that this is often misinterpreted. While programming is rigorous, mathematical, and logical, the process of research is not so hard and fast.

Hoare: I think that’s right. In research one has to have a much freer imagination. In programming, after you’ve had the brilliant idea, there’s an awful lot of discipline, routine, and construction of code, which still requires great skill and great concentration. If you ask anybody who’s actually writing code, they will tell you that they have switched off their imagination. They don’t want to hear new ideas while writing code.

Frana: You said some moments ago that people have been critical because you've emphasized the mathematical. Do you see that you've emphasized mathematics to the exclusion of other things?

Hoare: I hope I haven't excluded too many things. The students at Oxford and at Belfast had all had the usual opportunities. Indeed, we set tasks that they have to perform in the practical laboratory—actually writing programs and solving problems. It's the challenge of the educator, I think, to put each practical exercise in a theoretical context so that it illustrates an idea of greater generality than the particular program that is being written. It's like other examples; you have to give examples, but you also have to make the student see them not just as a set of examples that have to be learned, but as illustrations of a general principle that can be applied to any other examples as well.

Frana: And this is something you learned at The Queen's University and then went out and applied at Oxford?

Hoare: Well, I did appeal explicitly to the analogy of a physical or chemical experiment, which should also have the same property. I used to draw this analogy that many students would regard their training in computing as being nothing but a lot of experiments—like stinks in the chemistry lab. Their teachers really have to try and emphasize the formulae and get people to see an experiment as verifying a formula rather than just making a pretty colored liquid.

Frana: Right. So I take it that this means that the students are in a hurry?

Hoare: Yes, students are like other people. They are trying to optimize their concerns. If you told them what the exam question was going to be they wouldn’t waste their time on lectures. I quickly learned that the characteristic of a good course lasting one term is that after the first half of the course, the students just can’t see what it is about at all, but at the end of the course they can’t see what they found difficult anymore. So you have to choose a fixed time frame, and if you don’t have that initial confusion and doubt you’re probably not teaching things that are stretching your students enough.

Frana: That’s very helpful. Were you appointed to the James Martin Chair then when you went to Oxford?

Hoare: The Chair at Oxford was only a recent creation when I arrived. It was created for my immediate predecessor Christopher Strachey, who is also a big name as a pioneer in the theory of programming. He died quite young. Oxford established the chair, and it then went by the original name that Strachey had given it as a chair in Computation. The name was changed by the generosity of James Martin, who made a donation to the university to fund a chair in his name.

Frana: Belfast was a very difficult place to be in the 1970s wasn’t it?

Hoare: Yes. I moved to Belfast with my young family just two weeks before the first disturbances in Londonderry. We lived through the thick of the troubles in the early 1970s. 1972 was about the worst and most depressing year.

Frana: Which they've only just apologized for yesterday.

Hoare: Yes, I know. That took me by surprise too. I still take an interest in and sympathize with my colleagues and friends over in Northern Ireland. In fact, I visited them earlier this year and had a very pleasant time.

Frana: I've talked to many people in the software development community, high and low, about you to ascertain the popular view on your work. What they tell me over and over again is that at some point in the late 1980s or early 1990s you softened your position on Industry adopting some of the theoretical trappings of computer science. When I read the literature I don't get that perception at all. I found you writing about how the gap is tolerable, but let's work to close it, or keep things going along in parallel.

Hoare: I like your second interpretation. You can't build bridges by just concentrating on one side of a divide. You have to see the problems that arise on the other side as well, but you can make your choices as to how much attention you want to give to them. I think that science and engineering and industry work well together if it's possible to recognize the different concerns and different time scales that the different players in the game are reporting to. I still feel glad to emphasize the duty, the defining characteristic of the pure scientist—probably to be found working in universities—who commit themselves absolutely to specialized goals, to seek the purest manifestation of any possible phenomenon that they are investigating, to create laboratories that are far more controlled than you would ever find in industry, and to ignore any constraints imposed by, as it were, realism. Further down the scale, people who understand and want to exploit results of basic science have to do a great deal more work to adapt and select the results, and combine the results from different sources, to produce something that is applicable, useful, and profitable on an acceptable time scale. I've always found that every time I look into a practical environment I find a new and fascinating theoretical problem. It won't always happen, of course, but research is most exciting when you suddenly find a new direction to run down. Where you'll find that new direction, I don't know. If you concentrate too much on the things that you're thoroughly familiar with, you will not find it. So just do the odd quixotic thing, go into your local bank and see how they're using computers, anything, but it should be quixotic, on the spur of the moment, at random, and then think about it. It's essentially taking a walk amongst the daffodils, as it were.

Frana: Another the comment you make that seems to come up quite often—where you say, 'How did software get so reliable without proof?' A rhetorical question here, but can it be used to defend a lack of rigor?

Hoare: I think if you look in the article you'll see a number of reasons why industry really does best to concentrate on the results of completed research rather than research that is currently going on. The worst thing of all is to rely on the results of research that's only just started. And this has been such a pervasive mistake in the funding of research—a mistake which I myself have exploited—that the research initiative gets announced and funded on the basis of a political motivation, and then Industry is suppose to come in and assist in the work.

TAPE 2 (Side A)

But industrial engineers don't actually want to use new technology. It's always a risk. So you have the scientist, whose main motivation is to discover as much new technology as possible, and the engineer, whose primary motivation is to avoid using it as far as possible. They want to get away with something that is tried and tested. You'd be crazy to do something new, if you could avoid it.

Frana: Your Turing Award was awarded in 1980. Has there been a dramatic change in the direction of your research since that occasion?

Hoare: I think that the general themes remained close. By 1980, the major attempt to construct a fairly complete axiomatic definition of a sequential programming language was, well, in the bag. I adopted myself the challenge of applying the same techniques to concurrent programming. I was very explicit with myself about what I was aiming to do—to reproduce for concurrent programming the conspectus and the understanding that Dijkstra had given us in his book on the Discipline of Programming, and its technical underpinning in terms of weakest preconditions. It took my move to Oxford and collaboration with some very good students before I began to see how that goal could actually be achieved.

Frana: So there is continuity.

Hoare: Indeed, yes. And since then I've been trying to apply the same technique deeper into the theory and with different applications, different kinds of programming languages.

Frana: Now it's somewhat unusual that someone as deeply theoretical as yourself has had such connections to industry over the years. I had an occasion to talk to Dijkstra last summer in Austin, and he talked about the 'indecent industrial course work' that was often a part of the curriculum at Texas and elsewhere. Yet you seem to have found a way out of that conundrum, working first with IBM and now Microsoft. You have been granted rather free passage between these worlds. Is there a way out? Is there a way to integrate?

Hoare: I think it's a question of recognizing that there is a spectrum and choosing a point on it. I have been a programmer with industry, and so has Edsger actually. He worked for Burroughs for a number of years.

Frana: That's right. Consulting mainly.

Hoare: But Edsger is my model when I think of the pure scientist, and of the absolute importance that our society should support a good proportion of scientists working at the purest possible end of the subject. He quite rightly says that it is the attribute of the scientist to separate concerns, to make everything you possibly can irrelevant, and concentrate on a single theme at any one time. It is the task of the scientist to pursue absolute truth, absolute purity. He doesn't want to stop at 99.9 percent pure. He wants to go for bust. Edsger refuses to consider management concerns or even financial concerns. They are not the business of a scientist. I have a very high respect for people who can take into account those concerns, and still understand and apply the results of pure scientific research.

Frana: Well, your personal experience was very different. You had a very early management problem to overcome on the Mark II software system.

Hoare: Yes.

Frana: Did you read Fred Brooks' book *The Mythical Man-Month?*

Hoare: Of course Fred Brooks' problems came after mine, but yes I did indeed read the book. It just resonated, obviously: comforting that such things happen to other people. I remember after the failure of the Elliott operating system project, I read preliminary descriptions of OS/360, and I said to myself, 'Wow, these people must be very clever, how could they do it and me not?' And later I found out that they couldn't either.

Frana: Did you also add manpower to a late program, thereby making it later?

Hoare: Yes, I think I must have. It wasn't such a great amount of manpower.

[pause]

Frana: I know that this hasn't been a main thrust of your research—reasoning under uncertainty—but that you've been studying it off and on since your early education.

Hoare: Yes. I studied probability as a hobby at school, and I pursued it from a philosophical point of view as a means of explaining the phenomenon of uncertain reasoning. I actually spent a year studying statistics for a qualification at Oxford.

Frana: Was that with Leslie Fox then?

Hoare: No, the statistics course was with Norman Bailey. While I was on that course I attended a one-week programming course given by Leslie Fox.

Frana: I see.

Hoare: I think that since I've discovered computers and programming, I've sort of gone back to the original philosophical ideal of certainty, achieved apparently by mathematical reasoning, and concentrated very much on that aspect of computing. But I recognize that probability is becoming increasingly important in computing, both because it provides fast ways of getting solutions to otherwise intractable problems, and because when computers are connected together in a vast network—as they are today—a lot of things happen for which only probabilistic considerations can give us a basis for understanding.

Frana: Have you shied away from that in the past for a specific reason, or is it just not your specialty?

Hoare: It's just not my specialty.

Frana: Were you reading things like Leonard Savage—the idea of personal probability, subjective probabilities?

Hoare: On the whole I've been keeping away from considerations of imprecise reasoning. And because I think—this is perhaps some justification—it's a playground for charlatans. If I went back to uncertain reasoning, I would try desperately to make probabilities work because that is a very secure foundation for dealing with uncertainty.

Frana: If you quantify uncertainty then what have you got? Is that what you're reacting to?

Hoare: I think so, yes. Probability is a good abstract measure of uncertainty and it has a good calculus that backs it up, and that enables one to make predictions that can then be tested. I haven't otherwise found a convincing approach to a scientific study of uncertainty. I don't think any the worse of uncertainty for that reason. In fact, the more one goes into formality and mathematics and computers, the more you have to admire the way that people actually do it. I'd sort of given up on computer diagnosis in medicine, for example, as soon as I realized what your average doctor does when you come into the consulting room. He says to himself, 'I know what you've got,' and he hasn't asked a single question—something about the way you look, the way you walk, it's just indefinable.

Frana: What did you think of artificial intelligence?

Hoare: I did feel not very attracted to artificial intelligence in general. In some ways I've been right, and in some ways I've been wrong. I think that enormous progress has been made—toward artificial intelligence goals like, shall we say, automatic translation of languages—now that computers are so much faster and larger than they were. That's an interesting example because of its part in my own introduction to computers. When I was a graduate student at Moscow State University I got a letter from the National Physical Laboratory inviting me to apply for a post as a senior scientific officer to work on a project that they had for programming the Pilot Ace computer to translate from Russian into English. They'd heard of me as a Russian speaker and thought I might be useful to them. While I was in Moscow I spent a bit of time meeting people working on machine translation. I thought about it for a bit and came to the conclusion that it wasn't going to be possible. So I didn't pursue the offered job at the National Physical Laboratory. Instead I took a job with Elliot Brothers, where I had an opportunity to work on an artificial language, ALGOL 60, and write a translator for that, which fortunately did work.

Frana: Today these MT researchers appear to be interested in natural language processing. That is to say, what they are trying to do is take everything that's ever been translated and then sort of match it up. Is that how you understand it?

Hoare: I think the key turns out to be very different from anything that anybody was working on in those early days, when it was thought to be a question of algorithms. The way it turns out to work, and in other areas as well, is in terms of massive storage capability, the ability to store vast corpuses and then do statistical pattern matching in order to get an answer of increasingly reasonable quality. It really suggests that this may be the way the brain works, because learning and training by example seems much more intuitively reasonable than somehow executing an algorithm by program. That doesn't seem to be what the brain does at all.

Frana: You've been thinking about this lately then?

Hoare: On and off, yes.

Frana: Speaking of limited memory and execution speeds, have you read the thesis ‘The Evolution of Software Design’ by David Koepke, a student at the University of California at Northridge? In the preface he says that he contacted you and asked you a few questions. David attached, in an appendix, a letter he received from you. He concludes, ‘The limited memory and execution speed of computers of the 1950s and early 1960s forced programmers to concentrate on one design goal—efficiency. Programmers had to develop clever tricks, such as modifying their own code, to overcome the computer’s limitations. This restrictive hardware environment negatively impacted design and resulted in programs that were less clear.’ Would you agree?

Hoare: Oh certainly, yes. But I think that as programs became larger, the sensible programmer would confine the really tricky bits to rather small areas in the code, and seek to preserve structure and annotation in the vast majority of the code that is written.

Frana: One of the ideas that he imparts is the possibility of over-engineering the code. Is there also point where you can have almost too much memory?

Hoare: There are certainly diminishing returns, but the returns that don’t diminish are big enough that they make up for it. The phenomenon of legacy code is that nobody removes anything, and therefore, a bit like the genome actually, there’s probably a lot of junk in there, which nobody dares to remove because nobody knows whether somewhere, somehow it might be doing something useful. In the old days we used to use instructions from the code as constants if they happened to be the ones we wanted. Fetch down this instruction from store, mask off the bottom five bits, and use it.

Frana: Again, a quote from David Koepke's thesis: 'Programs are no longer viewed as sequential hierarchy of procedures (actions) that pass data up and down the hierarchy, but rather as a set of objects (data and actions) which remain in existence, execute concurrently if possible, and interact with each other by sending messages.' Does that encapsulate software design?

Hoare: I think it expresses a very, very important insight, yes. It mirrors the development that I made in my own scientific progress, in moving from sequential code hierarchy construction towards object-orientation and concurrency.

Frana: Is there a third shift that we're on the cusp of?

Hoare: You tell me?

Frana: I don't know. I think distributed programming is where we're at right now. I think this object-orientation is still quite highly regarded, and people are only now figuring it out to be honest.

Hoare: Yes, that's right. I think the advent of the Web and the possibility of really exploiting distributed computing is certainly considered to be a major new thrust and advance in Microsoft. Distributed computing and concurrency have been research topics for an incredibly long time. They've been quite well supported by the research funding agencies since the early 1980s. Everybody has seen it coming. It just takes an awfully long time to come. And it's certainly still not here. I think the practical uses of concurrency have still to explode.

Frana: Is computer science still a long way from possessing the framework that you and others have envisioned? McCarthy in particular has been very forthright about saying that we're perhaps farther away than ever. You don't feel that way?

Hoare: I think we can be making good progress, and the goal can still be receding. That's not impossible. Otherwise, the subject is dead when we catch up. What's happened is that practical programmers have used whatever understanding they've gained, or theoreticians have offered them, to increase complexity. So programming languages on the whole are very much more complicated than they used to be: object orientation, inheritance, and other features are still not really being thought through from the point of view of a coherent and scientifically well-based discipline or a theory of correctness. My original postulate, which I have been pursuing as a scientist all my life, is that one uses the criteria of correctness as a means of converging on a decent programming language design—one which doesn't set traps for its users, and ones in which the different components of the program correspond clearly to different components of its specification, so you can reason compositionally about it. Programming languages in full generality have not really paid full attention to this aspect. They tend to be defined purely on the basis of what the machine does when it's executing the program, and often at quite a low level of granularity too—individual storage accesses. I still think we have to get the message across. The tools, including the compiler, have to be based on some theory of what it means to write a correct program. That message is one that I keep preaching. Microsoft is a good place to preach it.

Frana: You've also warned that there may be disastrous consequences. Have you ever testified in a criminal or civil trial where software may have failed?

Hoare: No. I've fortunately been spared that, because I think if one were giving testimony one would have to admit that the state of the art as generally understood does not include the use of proof techniques in programming. It could be argued to be unreasonable, and a court of law should enforce that. The law has to keep in step with public perception, public acceptance, and means of detection—it has to be realistic. I've stopped trying to make people's flesh creep by telling dreadful stories of what might happen if software goes wrong. I think it's ineffective. I think that software has gone wrong enough to make people realize that it is expensive to make mistakes. Some of the recent viruses have been extremely expensive. The trouble is that when things go wrong there is always more than one reason for it. There's always more than one way in which this disaster could have been prevented. Ultimately, you can always say, 'If only our test regime had included the test which revealed this error, it would never have gotten into production.'

TAPE 3 (Side A)

Hoare: There are other measures that one can take to reduce the likelihood of such errors. There are easier ones than starting to use formal proof in verification of software. You can’t blame practical people for doing the easier things first. So I tend to go further back than I did when writing about how did software get so reliable. I regard the theory as something on which you base tools, which can be used to prove the reliability of software. It’s not going to be used directly by the programmers except through the medium of the tools that help them to use it.

When I was an academic I deplored this outcome. As an academic and teacher you want people to understand things and apply them. The prospect of getting it all done by the computer would seem to be a cop-out. I don’t feel that way any more. I think actually if you take the analogy with other areas of engineering, and increasingly of science and even mathematics, you can see people do not have to learn the vast number of formulae they used to learn. Instead, they have to learn to use the computer effectively. This frees them, I feel, to understand concepts and the foundations while they’re learning the mechanics of the application of the theory. I think no engineer does his own structural analysis any more. Engineers have to understand the concepts of structural analysis in order to be able to use the computer effectively. They have to understand it more if they want to make innovations, not just be limited by the tools. Until we have the tools, the compilers, and maybe even the languages that will embody the theories and make them applicable, I think programmers will have to do the best they can with their intuition and their education as it is.

Frana: Those are really remarkable comments. I don't think people who know you only from your 'pre-Turing' days would expect you to say such a thing—

Hoare: I've disappointed people's expectations. I have to say it's quite a common phenomenon. I would like to emphasize that I don't regard this as invalidating the enthusiasms, and even the misguided enthusiasms, that I had when I was younger. I want young people these days to be equally idealistic and enthusiastic. I hope many of them will be naturally enthusiastic. Unfortunately, many people tend to feel, as I did, that the value of their work was being downgraded by the fact that it wasn't being applied. Many people weren't willing to try to apply it, or even believe that one day it would be applicable. What I would like to do now is to tell people that this is not so. Your work is essential. It is going to be applied. This is all future tense, but certainly Microsoft needs it. They need to be able to control this phenomenon of error, particularly when we move into concurrency. You've got to keep researching this stuff and you've got to keep teaching it, because we need people to understand it. And these are the people who are going to lead the profession in the future.

Frana: So, again, you're facilitating a situation similar to the one in the past you set up, when you said to yourself: 'I know what I'm doing now will not see the light of day in industry for thirty years, but it will see the light of day.’ You’re advising young researchers to approach things much the same way.

Hoare: Yes.

[pause]

Frana: A fourth reason I am here is the historical interpretation of code. Conserving our software heritage is something of great interest in the professional history of computing. There has been a great debate, actually, extending back a decade or more between those who see the actual code as something we should work very earnestly to preserve, much in the same way that we’ve preserved some of the hardware. There are others on the far side of the fence who say that this is a tremendous waste of resources. It would be wonderful to preserve code, but this being a world of limited resources, we should look at doing other things. Perhaps, these people say, it’s more important to have a videotape of someone using a software system than to actually preserve the underlying code.

Interestingly enough, at this conference I attended up at the University of Strathclyde, there was a keynote address by Wendy Chun. She said that software is ideology, and that all ideologies can be reflected in software. She said, quite eloquently I might add, that ‘both software and ideology try to map the immaterial into material terms.’ But it is still very hard to convince historians to preserve these things.

Hoare: Well what are the costs of these collections? I mean the number of bits involved is minuscule.

Frana: I think historians are thinking instead about the opportunity costs of historians actually accessing this kind of material.

Hoare: Well, that’s really true. One has to envision a situation in which the archaeologist of the future wants to find out how they actually managed to get anything useful at all out of those early machines. It is inconceivable that the first computer I worked on cost in modern currency values a half a million British pounds. It worked at two thousand operations a second and it had twenty thousand bytes of storage. It will be inconceivable how we used these machines, just the same way that now we don’t know how they built pyramids. So can’t you imagine the television program of the future wondering, how is it done? And if you can imagine that, then I think as an archivist you have to keep the code. There’s something amazing about Dijkstra’s THE System. There’s something amazing perhaps about the early programs that we wrote. They are so tiny. How did I do it? I did some very—well, so did all of us—some quite clever things to get things as small as they were. For example, in order to get forward jumps working correctly, our early ALGOL compiler read the intermediate code backwards from its output on paper tape. That technology is quite irrelevant now, but it was pretty fantastic then. I think you should be thinking carefully about what bits of software, or representative samples, that you are going to keep, or you’ll be accused of blowing up the archaeological remains!

Frana: I don’t think anyone is minimizing the importance of the code. It may be, in my view, that historians are simply not programmers. Perhaps a stumbling block is this idea that you need to be able to emulate the code.

Hoare: That’s no problem. You would have to keep the machine descriptions as well. In those days machines were fairly simple, and people put a lot of effort into making the descriptions complete. It’s an undergraduate project to write an emulator for a machine like that. You can certainly get working copies that will go on working into the indefinite future. I think the Computer Conservation Society is coming to realize that software is the durable intellectual legacy, and the hardware has to be allowed to decay. Dijkstra would agree with that.

Frana: Oh, I’m sure he would. The reason I ask you about all of this is because in your piece, ‘Theories of Programming,’ you write about software as a man-made artifact. You take a very anthropological, human-centered—almost cultural—view of software. You argue that it is just like anything else that humanity produces, like art or architecture or literature. I’m wondering if there are obvious places where we can show how software is a reflection of society and culture or where culture has somehow impinged itself on the software. Are there examples that come to mind for you?

Hoare: Yes. I’ve occasionally thought of political and sociological analogies of operating systems trying to manage users or user tasks in a manner which is equitable, or at least not subject to abuse. There’s a whole analogy which I felt could be perceived very far. I suppose the artificial intelligence movement is one that has had quite a big cultural impact. The analogy between the mind and the machine really affected the way people think about minds—probably incorrectly. But it was, and to some extent still is, the analogy of mind as a mechanism that people tend to think of. It is comparable to the Newtonian discovery of the force of gravity in affecting the way people think about material objects in the real world. The ideas of scientific determinacy, for example, were inspired by the successes of astronomical predictions, and are really quite pervasive in the West. It's really quite an unrealistic expectation, but nevertheless it is part of a shared culture that you do believe that science, in principle, can predict a great deal more than in practice it can. The artificial intelligence movement inspires the ideal that a computer can do what a human being can do, and it inspired some fruitful investigation into brains and behavior.

Frana: I think sociologists, in particular, have pushed the rest of us away from the analogy by exploring things like 'cat-ness.' Rather than comparing humans and machines, they've been comparing dogs and human beings, or cats and human beings. If you can't somehow put those in the same category, they argue, how do you expect to bring a machine in and put it in the same category.

Hoare: Well the strange paradox of an artificial intelligence is that it turned out to be quite easy to do what people think of as being highly intelligent, like playing chess, while what's almost impossible to model is the interaction of a squid when it sees a threat. It seems like the squid somehow can do things that computers just have no idea how to do.

Frana: The other side of the coin is: how do we bring the humanities and computer science closer together? I notice that you have always encouraged students of languages to enroll in computer science. I would assume that goes for philosophers and historians as well, though there are very few of these people on the staffs of major software companies like Microsoft.

Hoare: Well, obviously now there are now so many computer science graduates that they are going to outnumber other disciplines in the best computing departments. If you take absolute numbers, you probably have as many philosophers now as you ever had. I think one has to regard this as exceptional. It didn't take long teaching an introductory programming course in Belfast to realize that in a class of forty there were going to be four, or maybe even more, who would never, ever write a program. They were just psychologically incapable of writing a program. One of them was left-right mirror blind. He could not distinguish left from right. To teach him that the variable that you want to change has to be written on the left of the equal sign and the expression to be written on the right, was a hopeless task. The person could never write a program. But he might have been a very good poet, very good philosopher, historian, or whatever.

Frana: I'm reminded of the Roman Catholic Church. The Church often looks for priests in strange places. They like to find people with 'unusual' backgrounds—maybe they look for computer scientists—and then make them priests who can go out and explain things in a way that appeals to computer scientists. Computer science also seems to prize nontraditional routes for acquiring expertise. There are all kinds of stereotypes of hackers coming into the field with a rather bizarre set of past experiences. Has the time passed where that is possible? You mentioned the advantages of looking at bank operations on the spur-of-the-moment.

Hoare: Well, as a scientist, I was suggesting that one should occasionally take quixotic looks to experience. As a person I think this is very rewarding. As far as programmers are concerned, as a teacher of programming, a researcher and a theorist, I hope we are beginning to build up a body of knowledge that will be recognized as essential grounding for people in the field. On the other hand, as you point out, there are still plenty of scope for the untutored genius, let's call it. Or 'hacker,' you may call it. They can somehow see the way the computer is going to work and just go for it. They have a role to play. I think you'd find a high proportion of such people in computer games companies. But then, you see, games is obviously an area where you need some kind of artistic flair to get in to the big time. For team projects, for teams working on legacy products, one would expect that a thorough grounding in the basics of the subject might be more useful. You want to attract people who are perhaps less gung-ho, more meticulous, more disciplined, sharing a cultural heritage with their colleagues. This is still not the case. In Microsoft and elsewhere, it's perfectly reasonable for unseasoned outsiders to come and make a very rapid contribution to the product and to the programs. I don't wish to be exclusive. On the other hand it does challenge our feeling of satisfaction of achievement when such untutored people feel that what we teach isn't really necessary.

Frana: And why is that?

Hoare: Well, I think what's happened in other disciplines is that it becomes compulsory, whether it's necessary or not. Doctors have to learn the Latin names of the bones in the body or perhaps you can't get your license to practice. The exam is the thing that defines your competence. The exam hopefully is highly correlated with competence: your competence to talk to your colleagues, and your competence to keep up with developments in the field. Now those are two competences that are conveyed by education. They aren't available to the untutored genius, who falls down when faced with complex problems where those other abilities are necessary.

Frana: Do you think computer science needs a very pragmatic set of exams?

Hoare: My mind's divided: I like the excitement of doing new things. I like the fact that people can come in with new ideas. I deplore the fact that so few people really do have a theoretical underpinning that in other professions is made compulsory. I fear that our subject will be more boring when things are standardized.

Frana: And those studying legacy code fit in—

Hoare: Yes, it's not as interesting. You have to get your satisfaction from other things besides, 'Gee, look what I made the computer do!' You've got to begin to get your satisfaction from writing tidy well-documented code independent of whether it's actually faster or more efficient or more reliable than ragged, lousy code. I think a computer science education can be made interesting and fascinating. I’ve met many lecturers who can. You can interest not only undergraduate students, but you can increasingly interest professional programmers in pursuing their subject from an academic point of view, from a scientific point of view. One of the reasons I moved from Belfast to Oxford was to try to establish this principle: you can teach interesting and useful theoretical computer science to experienced programmers with five to fifteen years of experience. I realized that in Belfast the population wasn’t large enough to run a Master’s course like that. A few years before I retired we got an opportunity to start up such a course at Oxford, and had some very good teachers who’d been doing it for a number of years. They’ve been doing it with enormous success.

TAPE 4 (Side A)

Hoare: Programmers from IBM and many other companies have gone through and obtained Master’s degrees in software engineering because of this Institute.

[pause]

Frana: Do you have a well-defined group here that you work with?

Hoare: Presently, I’ve gotten myself into a role of research consultant rather than an active researcher. I taught quite a bit to researchers in Cambridge who are working on theoretical topics or using theoretical methods to work on practical topics. I also taught to developers and researchers in Redmond. I think it's important I shouldn't try to set myself up in competition with them, and that I should retain a wide view of what's going on. Occasionally, one can put somebody else in touch with somebody else who's working on something that is relevant.

Frana: And that is the role of a Senior Researcher at Microsoft?

Hoare: Yes. It's a role which I heard about when I was visiting the Cambridge Research Laboratory a couple of years ago—just before I joined. In fact it was this talk that inspired me to sign on the dotted line. Yes, there's still much to do as one becomes older. Computing has lacked its senior citizens, as it were, and it's a role that people can fill without feeling the need to retain quite such an on-the-ball cleverness as the young people who are still very good at.

Frana: Are there any connections between Oxford and Cambridge? Not true?

Hoare: Computing departments in Oxford and Cambridge have very different approaches to everything. On the whole universities keep to themselves. University departments don't link up with other departments in the same subject, in any very strong way. Until Oxford recruits people from Cambridge, and vice versa, we will see a different culture in the two places.

Frana: What is your five-year plan for the future?

Hoare: I think trying to improve the morale of academic computer science is very important.

Frana: Which you perceive is, I gather, fairly low at this point?

Hoare: I have felt it was low, at least in Britain, at about the time that I was retiring. Microsoft is taking a long-term view of research. I think we are in a good position to encourage universities to take an even longer-term view. Another thing the senior citizen can do is to propose challenges to identify interesting research areas, areas where perhaps a little bit of collaboration will produce long-term results on a larger scale.

Frana: Are you willing to share the specifics of any of these challenges?

Hoare: Oh yes. My favorite challenge is called the ‘verifying compiler.’ It is a compiler that takes a program, together with its annotations and specifications, and uses mathematical techniques to ensure the one is consistent with the other.

Frana: You have written a bit about that.

Hoare: Yes. McCarthy originally proposed the idea in the early 1960s. Later on the first steps towards it were taken by Bob Floyd when he wrote about assigning meanings to programs. Maybe we’re now at last in a state in which one can begin to think of this as a realistic goal. I know the people in the world who might work together, and agree to use the word ‘verifying compiler’ to describe a collaborationist objective. I think it’s important to encourage other areas where a grand challenge of this kind might be productive.

Frana: Finally, may I ask about your wife Jill? How is she and is she continuing her interests? Is she somehow affiliated with Microsoft as well?

Hoare: No, no. Recently we’ve bought her a computer and she’s enjoying sending e-mail and occasionally looking up things on the Net, and storing photographs, and a catalog of garden plants.

Frana: And you are a gardener as well?

Hoare: I do some of the digging. We have a large garden, so in fact I pay people to do most of the digging too. I mow the lawn, and I enjoy it.

Frana: Thank you so much Tony. I’ve come to the end of my list of questions. I appreciate your time.

Hoare: Excellent.

END OF INTERVIEW