Donald Knuth: All questions answered University of Oslo, 30 August 2002

Tom Lyche: It is a great pleasure for me to introduce Professor Donald Knuth. Professor Knuth is a Professor Emeritus in The Art of Computer Programming at Stanford University. He is one of the leading researchers in computer science, and has made many fundamental contributions in many areas, including combinatorial algorithms and discrete mathematics. His monumental books, *The Art of Computer Programming*, have been seminal in computer science, and his typographical system TEX is used heavily, especially in journals requiring mathematical typography.

He has many honors. He is a member of three national academies in the United States, and he is a foreign associate of the French, Norwegian and Bavarian science academies. He has received the Turing Award, the National Medal of Science, the John von Neumann Medal, the Steele Prize, the Adelsköld Medal, and the Kyoto Prize from Japan, moreover the Harvey Prize, ..., I hope I get them all. He holds honorary doctorates from Oxford University, the University of Paris, the Royal Institute of Technology in Stockholm, the University of St. Petersburg, the University of Marne-la-Valée, Masaryk University, St. Andrews University, Athens University of Economics and Business, the University of Tübingen, and from Monday, also from the University of Oslo. [applause]

Professor Knuth has a long-lasting and close relationship to Norway. In '67 he came to an IFIP conference in Oslo where, among other things, SIMULA67 was presented. He spent the academic year '72–73 at the Univerity of Oslo, and this visit was influential for further development of computer science in Norway.

So, a little bit now about this session. The format of this session is informal. Anybody who has questions will ask them directly to Professor Knuth; also, those who had sent me questions before can pose them directly to Professor Knuth, and he will repeat the question for the recording. And I'll start by asking a question myself. [laughter]

When you came here in 1972, you gave a proof that Norway should not become a member of the European Union. [laughter]

DEK: yeah, yeah, yeah ...

Lyche: Is this proof still valid?

DEK: [laughs] Okay, very good question, Tom. Yes, I came, and I gave a tongue-in-cheek lecture — it was my first lecture at the University — and I didn't realize that it was something that you shouldn't joke about, because as I rode the trikk[1] back to my apartment, I noticed that I was moving from "Stem nei" to "Stem ja"[2] country.

The idea of the lecture was really mathematical. It's interesting to economists, the study of a three-way duel, where there are three players. And in my presentation, if I remember correctly from 1972, there were three players; they were named Petersen, Quisling, and Rasmussen, and they had probability p, q, and r that when they fire a gun, they would hit the person they shoot at. So I worked out the theory of what's the probability of survival as a function of p, q, and r. And the answer was that the one who had the smallest probability had the best chance of living, that the big powers were shooting each other, and then the other one would be left. Anyway, that was my "proof", and it had nothing really to do with the Common Market. And I'm not sure I'm glad you brought it up or not. [laughter]

It does, in fact, seem to happen only with three players; the phenomenon doesn't occur with two or with four, and economists are still unsure about this, but maybe that's the reason ... In English we have a word "truce", and it seems to start with "tr", which is the prefix for "three". There's this uneasy truce, where actually all players do best by firing in the air, every time it's their turn to fire. But the result doesn't seem to be true for other numbers.[3]

We're trying here to get the connection to the Internet going, so that I can show you my home page. Before I start with more questions, I just want to say how much "jeg elsker dette landet."[4] [laughter] My ancestors came from Schleswig-Holstein, which is a little bit close to Scandinavia, but my academic ancestors are almost entirely Norwegian. The first time I came to Norway, in 1967, I fell in love with the country and decided I wanted to be here a lot. My academic ancestors ... my thesis

This Q&A session was held in conjunction with the celebration of the bicentennial of the birth of Niels Henrik Abel, as well as the 25th anniversary of Oslo University's computer science department.

Thanks to Dag Langmyhr for providing a copy of the recording and especially for correcting the Norwegian references.

The video can be accessed at `http://www.ifiuio.no/aktuelt/arkiv/2002/09/allquestions.html`.

[1] electric tram

[2] "Vote no", "Vote yes"; this refers to the 1972 referendum on whether to join the European Union.

[3] *Journal of Recreational Mathematics* **6** (1973), 1–7.

[4] "I love this country" — a play on the Norwegian national anthem, which begins, "Yes, we love this country."

adviser was an American, but his adviser was Øystein Ore, at Yale University, and Ore was a student of [Thoralf] Skolem, Skolem was a student of [Axel] Thue, Thue was a student of Elling Holst, who was a student of Sophus Lie, and Sophus Lie was a student of [Carl] Bjerknes, and next week I'm going to find out who Bjerknes was a student of,[5] and so on. [laughter] But anyway, in that sense I'm a son of Norway.

And pretty soon we're going to be able to see my home page ...

The other thing I should say before we start out is that I want to pay respect to Ole-Johan Dahl. We had, of course, a special memorial session for him this morning, and both Dahl and [Kristen] Nygaard are extremely important in computer science. I had a very close relationship with Ole-Johan; he's one of the five people who had the strongest influence on my whole life. We were very close personally, and I invited him to Stanford for a year after he had invited me to Norway for a year. I looked at my diary before coming here, and found out that on my previous trip to Norway, which was 1994, I spent five nights at his house playing piano with him.

[laughter; DEK looks at screen at the front of the room, which shows a regular television channel.]

So we have a real hacker here getting us to the outside world. Well this is great — maybe I can get a copy of this video.

When this building was dedicated, Ole-Johan and I played the piano in the lounge, and we ... [looks at screen again] X-watch — all I want is Netscape!

Eskil Brun (AV technician)**:** I'll log in as me, in case you don't remember your password. You only got your account yesterday.

DEK: This is true. You don't have to log in. Just get me Netscape, and I'll destroy your files.

Brun: It's only 4.75.

DEK: The reason I wanted to know is if I can use Ghostview if necessary.

Let's get to my home page. Let's just see what you've got here under bookmarks. You don't have any bookmarks. [laughter] [up comes Google on the screen, then DEK's home page[6]]

Today my subject is called "All questions answered", and it's based on an idea that was started at Caltech when I was on the faculty there. It was started by Professor Feynman. At the end of all his physics classes, the last day of class was optional; students didn't have to come if they didn't want to, but if they came, he would answer any questions they had, on any subject except religion or politics. I liked that idea too, and I kept it up when I went to Stanford. If any of you ever were there in one of my classes, you'll know that the last day of my class was always called "All questions answered" on anything except religion or politics. Or the final exam — that was the other excluded thing.

Now today I already talked about politics, and it was a disaster. [laughter] I'll talk about religion only if there's a strong feeling for it, but basically, I want to answer any question that anyone in this room wants to ask — except the "Frequently asked questions", because you can always look up the answers on my home page. So ask me an "Unfrequently asked question" preferably. [looks at monitor] What else have we got here ... Recent news ... I was a million years old at the beginning of the year [Web page for $1\,000\,000_2$ year Knuthfest[7] pops onto screen] , if you use binary notation. And there are other things that are on this page — you can explore these things to your heart's content.

This page[8] shows all the books that I've got out there, with errors in them, and anybody who finds an error the first time, I really want to know about it. Then I write you a check if you're the first one, and if I believe that it's an error. And unfortunately, it usually is. [laughter] But the way I write books, I try to maximize my chance for error. I mean, a book is more useful when there are more ways that it could have been wrong. And so, instead of saying that something is better, I say, "oh, it's 12.8% better." And maybe it really is 12.7% better, so that's an error. So there's many chances all the way through to be wrong, and I try to get it right the first time, but the fact is, when I was a college student, I did not get 100% on all of my exams. Sometimes I would get 99, you know. Now, with several hundred chances to make an error on every page, you can see how many errors there probably are in my books. I'm trying to get them fixed. And also the errors in software — there's a rumor that somebody found the first error in TEX since 1994, and if so, I have to pay $327.68, which is an amount that kept doubling until it reached 2^{15}, and then I froze it at that point. [laughter] So anyway, now I'm ready for real questions. If you ask in Norwegian, somebody here will translate, so don't be afraid.

[5] The list is given on DEK's web page. Bjerknes was a student of Bernt Holmboe, who was a student of Søren Rasmussen, who was self-taught.

[6] http://www-cs-faculty.stanford.edu/~knuth/

[7] http://forum.stanford.edu/events/knuthfest02.html

[8] http://www-cs-faculty.stanford.edu/~knuth/books.html

[Q]: [in Norwegian; translated]

DEK: Do I think Sweden is a better country than Norway?

No, but it's interesting.

They have some good ... What was the name of the ... I was a fan of one of the skiers. Goodness, I'm forgetting ... [some prompting] In the winter Olympics that were held in Lillehammer, there was ... I sort of fell in love with this one Swedish skier.[9] [more prompting and laughter] I can't even remember any more, so I ... but I never got to meet her. I did meet the king once; that was nice, and he said that his daughter likes computers. To me, when I was here, living in Norway, we went to all parts of the country, and it's certainly the greatest year in my life.

[Q]: [paraphrased by DEK] The question is about generic libraries, what is the role of little toy algorithms, instead of staying on the high level?

DEK: I think, in all fields, not only computer science, people can make the mistake of saying you should always stay on the high level. Everyone who is best at their field seems to forget the way they learned it, and they'll find out that some parts of the things they learned are more useful to them in later life. So then they'll say, I won't bother my students; they won't have to learn all the stuff I had to do when I started. And as a result, I think their students are missing a lot. And I think that even more so in computer science, it's very important to keep track of many levels at once. What was the word that your professor had for this? Goodness, now I can't even remember the name of the man who spoke about it,[10] but he had coined a new word, which sort of means "look at things at all levels". The professor on information design. [prompting from audience] "Polyscopic". It's more than a telescope — it's a polyscope. So you see, the people who are the best computer scientists have a certain kind of talent that is not very strong in the general population. I think one of the properties of this talent is the ability to shift levels, to see something in the small at the same time you're seeing it in the large — to know that in order to solve a big problem you want to add 1 to a little counter. And in order to add 1 to a counter, it's actually better to have it in a certain cache or something like this — to understand what's going on at many levels at once, and effortlessly to convert, to chain. So the high levels are great, but in principle, the more that part of our brain is also able, if necessary, to open the box and look under the covers and see what's there, the better we are. And the people who have that skill discover that it also correlates well with being able to make computers do tricks, and they can really resonate with computers. I mean, like Eskil here had to go back into a config file and change all kinds of settings on monitor and things like this, so he had to dive down into rather low levels of the system in order to get these pictures on the screen. That's not an isolated event. Every day, I'm still going to things that are at low level, even though I'm using high-level stuff all the time too. So I believe when you build a building, you start from the foundation and you build up. You don't start at the roof and build down.

I spent a lot of time five years ago designing a computer to replace the `MIX` computer in my books. My book, *The Art of Computer Programming*, when I started writing it — that was 1962, 40 years ago — I knew that I had to have a low-level machine in there, and I took all the machines that I could find in the world in 1962 and I found something that would be nicer than all of them but similar to all of them. I put that into the book, and now it's extremely obvious that's it's quite different from the machines of today. And machines, in fact, ... computers went through a period where they got to be horrible to program at the low level, because people stopped doing much assembly programming. When almost all the programs were being written by compilers, then people changed the design of the machines so that only compilers could enjoy writing programs. But when RISC computers came out, and I read the book of Hennessy and Patterson about ten years ago, I was so happy, because all of a sudden you could have a real ... today's computers were actually beautiful. You could look at the programs and you could enjoy seeing what the bits were actually doing in the low level. And so there was a window in time when machine language was nice again, although it will probably get bad before long. I mean, the Itanium seems to be a disaster from this point of view. But I was glad that I could find a computer for the present time and explain its low-level operation, and give an entire machine design that I could use instead of `MIX`, and that's this `MMIX` machine.

If you look here [displays MMIX web page[11] on screen] this book is just the documentation of all the software that goes with it and makes it appear as if it was a real computer. It's just a textbook computer, but I had a lot of people helping me on the

[9] Pernilla Wiberg
[10] Dino Karabeg
[11] `http://www-cs-faculty.stanford.edu/~knuth/mmixware.html`

design of it — people who designed the MIPS chip and the Alpha chip both worked with me on MMIX. I believe that the future will prove that students who learn something about what's going on at this low level are going to turn out to be the ones who are going to be important in their careers, and the student who never learns about those levels and only learns how to apply somebody else's generic methods is not going to be of fundamental importance. The number of people who have the skill that I mentioned, of going through all these levels, is a small part of the population. The percentage seems to have been constant, as far as I know, at about 2% — one person out of 50 who's born seems to have this combination of abilities that makes them really resonate with computing. That's not enough people to solve all the problems that computers are able to solve. So those who have it should be sure to buy my books [laughter] and to use it. But the rest of the world, they should make friends with geeks like us, and then the other people can use these things. But just staying on the top level is as bad as a mathematician who only knows the statement of theorems and not the proof of theorems.

I gave too long an answer to that question, but it's something I feel rather strongly about, that you don't want to cut out either the low level or the high level.

[Q]: Is it true that you believe that it's not desirable to have bugs in programs? And what do you think about a new law in the United States that companies should be able to say that they don't take responsibility for any [problems]?

DEK: I didn't know about this law, but there have been a lot of worse laws passed, probably. As I said, politics — I have no talent for it whatsoever. I like to try to think, though, what is the best for the world eventually, and I came to the conclusion that it's almost impossible to get a program that doesn't have bugs in it. The reason is, I've never seen a program that exceeds a certain size that I could really say was bug-free. So we have to learn to live with programs that aren't 100% debugged, no matter how much I love to have programs that are bug-free. I would love to have TEX be one of the first examples of a program of more than a hundred lines — it's 15,000 lines of code or something is all — but I would love to say that this is one program that is absolutely solid. It hasn't been *proved* correct, but that doesn't mean anything. Well, it means something, but it doesn't mean that you're at the end, because there might be a mistake in your proof.

When I looked at the first published papers on proving correctness of programs, Tony Hoare's paper[12] on proving that the find program was correct, there were two bugs in it. He had proved it correct, but there were two bugs in his original proof. So I'm a strong believer that formal methods are helpful, but I don't ever want to say that now I've done this and I've got it right. You can prove the program meets its specifications, but how do you know the specifications are correct? It's almost as hard to write specifications as to write a program. So when you get to saying that something has no bugs in it, it seems to be impossible to get to that level. We could just say we want to get as close to that as humanly possible, or something. A friend of mine works for the government — he's the head of all the software that goes into air traffic control; there's a team, but he's the leader of it. He's based in California — he's being mandated by Congress that these programs have to be completely bug-free. And the Senators don't want to believe that this is impossible, so they'll only listen to the people who tell them it's possible, and the people who tell them it's possible just are saying this because they know that it'll pay their salary.

I wrote a paper called "The Errors of TEX",[13] which was the entire history of the debugging of this small-scale program that I wrote for typesetting. The beginning of the paper said, "I make mistakes. I always have made mistakes and I probably always will." I have to learn how to live with a life where I'll never be sure that I've got it right, but still get better and better.

Another friend of mine works for what we call BART around San Francisco; it's the subway system. They're building a new extension that goes to the San Francisco airport. And his boss is requiring him to have no bugs in this software. I don't know ... you've seen the movies, you hear the message that says "This airplane is being flown completely automatically, but don't worry, it can't go wrong, can't go wrong, can't go wrong..." The thing is, Norway has a history of innovative ways to improve the ... The man (wasn't he in Bergen?) 20 years ago who introduced "bebugging",[14] where he would say I have a program, and you want to see how robust it is, so you introduce random errors in it and see actually how long it takes before anybody notices. As a way to get some feeling as to how well you've checked out a program — I can't remember the name of the product — but anyway, that was quite influential.

[12] *Communications of the ACM* **14** (1971), 39–45.
[13] *Software — Practice & Experience* **19** (1989), 607–685.
[14] Tom Gilb, *Software Metrics* (1967).

So anyway, I think if lawyers have to get into it, they should have some way so that an insurance company can't say it doesn't have to take a risk for being sued that somebody's saying, "Well, my child died because of a bug in this program, and you're at fault." There's a certain level of care that is reasonable to expect, but another level that I would say is unreasonable. So it's not necessarily impossible to have some reasonable law around the problem that you describe. But I would say the chances that we've found the right balance at the beginning are very low. I don't know the details of the law.

Okay, good questions.

[Q]: Relating to a problem in the Norwegian government where there are a lot of legacy systems, where they want to have them cooperate and work together, is it better to start over from scratch instead of trying to run the programs that were patched together over the years?

DEK: In my personal experience, every time I started over from scratch I was happier afterward that I did it. In fact, TEX I scrapped entirely. After five years I took everything I learned and said, okay, let me start over again, and I'm *not* going to try to be compatible with the other. If this system is going to be important, it won't be more than a year or two before there will be more users of the new system than ever care about the old system, never heard about it. So no matter how much time had been invested in the other, it would be a small percentage compared to how much better the new one would be. I know many, many anecdotes over the years where this is true, and very few of the opposite, where preserving the past turned out to be successful. And part of is because of this problem of bugs. Every time you look at the old programs, you see that they don't really do what you thought they were doing.

Ken Thompson[15] told me, at Bell Labs he went through one of Bell Labs' most important applications that had been built up over a period of years — I don't remember if it was something about how they charge for telephone lines, or what it was — but he took this program and took a look at it and within one month he had identified several serious bugs and he had also been able to make it much smaller, more reliable, give more flexibility to it, and so on.

When I was a college student, one of my first jobs was to ... There was a company in Cleveland, Ohio, that made what they called bearings; it's part of the motor of a car. This company had what they called a "load study" program that would study ... the engineers would put in the design of one of the bearings and they would simulate on the machine whether it would be strong enough to take the pressure of the thing. This was one of the leading manufacturing companies in the United States, and they had been using this program for some years. I was hired by Burroughs Corporation, who wanted to sell a Burroughs machine, because the engineers had an IBM computer before. So I took the program that was running on their IBM computer, and all I had to do was convert it to the Burroughs language. I thought, okay, it paid $200 or $300, I can make some money, I'm a student; but I didn't realize that the Burroughs computer didn't have floating point subroutines, so I had to take a month to write programs to calculate arctangent and everything else that Burroughs didn't have in its library. I finally got to the point that I could take the program from the IBM and put it into the Burroughs machine; and I *didn't* get the same answer as they did.

I figured out how their program worked, and I made it run ... supposedly it was running faster; instead of being able to have four parameters, I was able to give it ten parameters; and so on. But I didn't get the same answer. I was going nuts about it, so I finally found a place that had an IBM machine, and it traced the program line by line, through thousands of instructions, and printed it out on an old line printer — nowadays you can't imagine what people had to cope with back then — and I compared the intermediate results with my program on the Burroughs machine, until I finally got to a point in the middle of the thing where the engineers' IBM program was overlaying some of their floating point data with a machine language instruction. I hadn't realized this conflict of the code. So what happened was that they were ... if I had to get the same answer as their program, I would have to clobber a Burroughs floating point number with some IBM machine language instruction. [laughter] So I had to come to the engineers and say, "Well, you know, do you realize that your answers have been more than 10% off all the time you've been using this program? And wouldn't you prefer to have the correct answer?" [laughter] I should have charged another $50 probably for this [laughter] but this was just my first experience of many where, every time I took an existing program, I found that it was less work and a better result — you know, total work — to redesign it and to not ... Now, you still have to deal with the problem of old data files, and being able to deal with different formats, but that's as far as I would go, and I would get out of the old data format as soon as possible.

[15] In the talk, DEK mistakenly said Ken Knowlton.

[Q]: How does "second system syndrome" impact on the answer to this question?

DEK: You must define for me "second system syndrome", which sounds like a great thing I should have known. [laughter]

[Q]: It's something that Frederick Brooks wrote in the *Mythical Man Month*.[16] The "second system syndrome" is the tendency of the person who has written a system once and is trying to rewrite an equivalent system a second time, to try to do all the things that he didn't do in the first one, so Brooks's theory is that the second system of one particular type that you write is almost always a disaster. The first one works, but is limited, and the second one is a disaster, and the third one, you learn from both the first and the second.

DEK: Okay. I knew that *Mythical Man Month* was sort of a "three-M", but I didn't realize that he also had his "three s"s in there. Of course, I read that book so I should have remembered. I have a great admiration for Fred Brooks, and he has lots of experience as a manager, which is orthogonal to mine, because my experience has been with small groups of people. I mean, to me, having a group as large as two — Dahl and Nygaard — is impossible; it's the only example I know in the whole world where you have two people with a strong personality complementing each other and working together to make a great product.

But it is certainly true that if you ... When I went, for example, to make the second version of TEX, I did not want to go much further than the first. It's very easy to be tempted, after you've understood one problem, to say okay now, that'll scale up, and in my new project I will be able to, now that I understand the small thing, now I know everything and so I can try a much bigger project, and my superior knowledge is going to make this bigger project really fly. So that's an important danger to watch, to extrapolate on your own knowledge. Like my `MMIX` computer, which I have here now, is my second system, actually. I started out with the design of `MMIX` ten years ago and it was a 32-bit computer; I realized it should be a 64-bit computer. But then I tried to hold back on things that I was putting into it just because they seemed to be cute. It's a strong tempation to do that, so I had to have a large focus group sort of keeping the control on.

Before I stop on this screen, by the way, I want to mention `MMIX` to people who are doing curriculum here. I would hope that somebody here would take a look at this, because it's become quite a ... There are several universities in Germany that are using it thoroughly in their teaching now, and a new book is coming out in German next month — introduction to computer programming, where they learn `MMIX` before they learn Java. They learn how to do simple programming, they get a feeling for how caches behave, and so on. Because in this computer you can design computers that we don't know yet how to build. You put in not only your program, but you put in a specification, a configuration of the machine. You say how many types of cache you have with different caching strategies, how many functional units you have, how much parallelism you're going to have, how many instructions are you going to execute simultaneously, and find out if that speeds up your programs very much. It's a metacomputer; it's a computer that has many parameters in it, and, well, John Hennessy said it was the cleanest design he ever saw of a machine language. I tried to make it so that it wasn't hard to learn and to keep in your mind. And the programs are kind of fun to write. So it's also a machine that I still think is five or ten years ahead of the state of the art, as far as actually building the chip for it. But the main idea was to make it of maximum use in the educational environment and for experimental purposes, so that people could play around with it. Now we've got a C compiler — it's in the `gcc` standard distribution now — to get code for it, as I understand, and a lot of good tools have been made for simulating it, and we had some experimental tools for visualizing the pipeline. It's something I recommend people here taking a look at, because I think it has use especially in pedagogy and learning things that are going to stay with people who make a career of computing.

[Q]: You commented that you haven't worked in many very large groups. But a lot of modern software development, especially in the open source world, is done by *very*, very large groups if a lot of people are contributing source, and basically [...] each other. How do you view the success of these projects, especially from the scale of the projects themselves.

DEK: So the question is about large projects. I mentioned that in my own case, I was the only coder of the TEX system. I was more restrictive in that sense. I mean, Linus Torvalds is still supposed to approve every line of code for the kernel, right? And I was even more of a filter than that. But people *would* suggest code. I'm not even sure how many

[16] Frederick P. Brooks, *The Mythical Man Month: Essays on Software Engineering* (Addison-Wesley, 1975). Second edition, 1995.

lines of code are in TEX and METAFONT compared to the Linux kernel, but he doesn't have to read through all the drivers, I'm sure.

But the question is, how does this work with the open source movement. Now, I've had nothing but great personal success in the dealings I've had with open source, but other cases where you have a large number of programmers working have tended to be less successful, in my experience. So somehow, open source is doing it better than all these other things that I had met in my life before. And certainly, when Fred Brooks wrote his book, he was reflecting on IBM's experience of trying just to throw more people at the job of writing software without realizing that this might make the project get worse instead of better. So there's something ... Part of it is the pattern that has developed for communicating, but I think still, in the open source world, you still have somebody who's the czar, you know. I met the guy who is gdb, and I met the man who's in charge of the C library. And, you know, I've met various people over the years who are really the gurus of individual parts of GNU Linux. So, my experience with open source has been, as I say, it was very good. I would find a problem in one of the programs; one of the examples was — oh, gosh, what's it called — rtf, the alternative to xterm, and it wasn't behaving the way it said in the man pages, and so on, and so I sent out a message. Within three hours, I had three answers from people who said one of my problems, oh yes, that's already been discovered, and it's fixed in our beta version if you want to try it. Pretty much all my experiences except for the GNU Pascal compiler have been a success. The GNU Pascal compiler has not yet been ... I decided to give up on that for the moment.

It's depressing if we have to come to the conclusion that no large-scale software efforts could be successful. But I found that developing a first-generation system, the fewer people, the better. A system like SIMULA couldn't have been developed — I just refuse to believe that it could have been developed by more than two people — the very first. After it's understood, then it's ready for a larger thing. But the closer something is to the source, and to being a breakthrough idea, the amount of bandwidth that you need to communicate what you're doing will easily swamp things. I can explain, for example, with TEX.

My very first draft of TEX I wrote in a long, ... I stayed up late, all night, and I wrote this thing, and I thought I had specified it completely. I gave it to two students, and they were going to implement it while I went to China for a month. So I came back from China, and they hadn't finished implementing TEX. I couldn't believe it — what's wrong? They had gotten a subset going, and they could typeset one line on one page, or something like that. But then my sabbatical year began, and immediately after starting to write the implementation myself I realized what the problem was. Because this was breaking new ground, this was a totally different system than had been before. And I found out as I started to write it, when I had written the specifications, I thought it was clear, but there's a million things you don't realize when you write in natural language; you don't understand that you're only answering a small part of the thing, and when you start writing code, then you realize all kind of things come up. So I started to write the system myself — I'm on my sabbatical — and instantly I discover, oh yeah, that question isn't answered. I have to make a decision. Now if my students ... So here's why a lot of projects have failed this way — the students, suppose I had farmed that out to them and I'm not on sabbatical. But I give it to the students, and I'm not in China, so they come to this point where they say, "Oh, what did he mean? What should I do there? Okay, I'll schedule an appointment with Professor Knuth, and I'll see him next Tuesday." So they'll come to my office, and then they'll start to explain the problem to me. And in fifteen minutes they'll tell me, and I'll say "Oh yeah, that's right. I meant to say this." So then I can send them back to work, and they can start again. And then, another five minutes later, they would come up with another question and would start to make another appointment with me. The thing is, these questions are coming up all the time, so if I'm the one who gets to ask the questions and answer them, I'm saving a factor of 10, 20, ... So a first-generation system really needs to have a small number of things, and the people, the lines of authority that they have to make decisions, and then publish them afterwards, has to be chopped up so that there's very little communication needed.

[Q]: In the U.S., you have laws like the DMCA Copyright Act and so on. These laws encourage not full disclosure [*sic*] on bugs and problems, and so on, and you have several researchers in several countries not especially wanting to say what kinds of bugs they have come into their programs. Tell me about these problems.

DEK: Everything I know about the Digital Millennium Copyright Act, it's apparently a total disaster from start to finish. I just hope that, you know, that we recover from it, and come to our senses.

One ray of hope was that they did scrap the initial idea for the Clipper chip and such things for encryption, after they thought about some of the more subtle issues. So what happened then? This Russian man found a very weak encryption in Adobe's book-reading program, or something like that, and then he got thrown in prison and certainly lost a lot of time and had to pay all kind of legal fees and so on — it was horrible. And even after Adobe said we made a mistake, we didn't mean to sue.

There are many parts of life where I like to say well, gosh, I'm so glad I'm only a scientist, and I don't have to think about how to interface with all the lawyers of the world. I had only once so far to do anything associated with the law, and it was because of the public key encryption. Apparently, since I was at Stanford University at the time, I could be forced to spend a lot of time being what they called "deposed".

But I have a standing offer for consulting fee with respect to legal issues. In my own life, I consider the thing that I do best is write *The Art of Computer Programming*. I've got at least 20 years of work to do on that, and if I'm lucky, I'll live another 20 years, so why should I do anything else? Well, I like to get an honorary doctorate, okay, so I come to Norway. But otherwise, I'd be home writing *The Art of Computer Programming*, and so if Bill Gates wants me to be a consultant to Microsoft, I say well, okay, but this costs $10 million a day, payable to Stanford University. But my fee for lawyers is $20 million a day. So you have to understand, I don't have this great connection with anything that has to do with suing people and things like that.

And I'm not a big fan of patents, either. I wrote an open letter to the U.S. Patent Office at Richard Stallman's request about ten years ago, and it was published with the CWEB manual; and basically I was saying, look, with software, if everything in the world is patented, then progress starts to go to zero, and almost none of the programs that are used every day would have even existed if they had been created in an environment of patenting. And I especially dislike patents on trivial ideas, that we would expect any student to come up with. I can understand patents for tough stuff, for difficult ideas. So when it comes to intellectual property, I see that the system is bad for reimbursing people. In some professions, people have to rely on getting paid because of royalties, because they aren't well compensated. As a scientist, I have a job where I can be paid, you know, to do research, by the government, because science is good for the community. But if I'm a musician, I can't be paid for practicing my music, or if I'm an author of novels, I'm not supported — a novelist isn't considered good for the community the way a scientist is. A font designer — we have a National Science Foundation; we don't have a National Font Foundation. So somebody who wants to design the things that we use all the time for reading, and do them better, has to rely on royalties somehow. I think that there are some flaws in the way people are compensated.

Then when we get to royalties, then it becomes one of these things where it's feast or famine. When you get a royalty for something, or you get a patent on something, then all of a sudden you're supposed to be rewarded to such an effect that you don't have to do anything more in your life — the money is supposed to come in for fifty years for an idea you had, for some work that you did in one year. So things are kind of mixed up in that area. I don't have the answers; all I know is that lots of stuff isn't fair and to me, it's better to pay people for what they do and the services they provide, instead of how lucky they were in doing something once.

I don't have all answers to all questions, I guess.

[Q]: What do you think about the Microsoft .Net strategy? Would you support it?

DEK: I don't know what the Microsoft .Net strategy is ... [applause]

I have to limit my ... I don't think I'll need to write about it in the *The Art of Computer Programming*. [laughter] As I say, I only have 20 years to go.

[Q]: What is your thought about quality in today's operating systems? Do you think there can be any huge development in the next 20 years, any breakthrough?

DEK: First of all, I have to say that the main reason why I'm happy with Linux is that my computer hasn't crashed. I haven't had to do anything special since last October, when I think it was because the electricity went down in the building, or something like that. It's very robust for the kind of things that I do. So I tell my friend — we were at a picnic a week ago — and the man's wife said, "okay, why aren't computers easier to use, and more reliable," and things like that. I said, "you should get Linux." "Oh! What's that?" And then we talked a little bit more, and she said, "but how am I going to use ... I have to use Intuit software" (which is a proprietary system I don't think works with Linux). I didn't have an answer for her on that.

But now you're talking really about the operating system, which lies underneath. I think ... I've never been involved with operating system research.

I think in MMIX probably the Achilles' heel at the moment is that with the machine design I have right now, it's not easy to make a virtual operating system, where somebody could have their own operating system and pretend to have authority to do privileged instructions. The machine, it's actually very similar to the Alpha design as far as the protection mechanism goes, but it's enough different that it might ..., anyway. I have a dream though, that if there's gonna ...

If a brand new breakthrough is going to come through in operating systems, I think it would be, it would probably come from a small group of people who start from scratch, and just say, let's re-examine all the old assumptions. Is it really true that what we should have is, you know, shared libraries of a certain kind, or something like that, when programs are running. What if we ... you know, all kind of assumptions. The operating systems we use today are based on the operating systems we used ten years ago, and those were based on the previous ones, and they're based on hardware assumptions, as to what is cheap and what is fast, and so on. So all the parameters are changing so rapidly, I think it would be worthwhile to rethink everything. For example, with MMIX I designed an assembler for it that is very primitive, so that anything can be assembled in one pass. And the assembler goes so fast that it competes with the loader. So what if you kept your program in source form instead of in binary? Because then you could have conditional instructions in there saying, you know, well I can allocate registers differently; I might give myself more global registers in this routine, depending on how many other routines have been loaded with it. All kind of things can be done, if your assembler is fast.

And so it might be a good idea to rethink the whole idea of an operating system, saying, well, let's say we don't have loading routines the way loading routines have worked for 40 years. And if you don't do that, then you probably aren't going to come up with major breakthroughs. Also if you do it, you might find out that, well, the old way is really better. But I think that would be an interesting thing for a group of people who have tenure, so that they don't have to risk their career on it, to spend five or six years just seeing what would happen if they could start over, all the way. Klaus Wirth did that fifteen years ago with his group in Zürich and Utah.

Other questions? Nobody's going to ask me about Volume 4?

Let me preempt the one question that I was thinking somebody would certainly ask. Let's see, here we go. [looks at monitor and starts to type] Oh, we don't have much here ... Anyway, let me just say something that's new, because I've actually been working this year on *The Art of Computer Programming* and getting pages out of Volume 4 that are ready for beta test. So anybody who's interested in seeing what might appear in Volume 4 someday is welcome to try looking at these pages. In fact, before coming here I also finished 40 more pages that I haven't announced yet, so if you can figure out how to get at them — there *is* a way — but the thing is, I've put here on my news page, I've put a list of exercises that nobody yet has commented on in the ones that I've put on line. I'm extremely grateful, hundreds of you have taken time to read these drafts, to detect and report the errors that you find, ... I'm getting ... the Internet is amazing! Within a week of posting these pages I had mail from all over the world; a fourteen-year-old boy in Nuremberg, Germany, wrote to me and told me I had spelled 'Nuremburg' wrong. [laughter] But some of the parts I worked the hardest on, nobody yet has commented on, so either I got it right the first time, or they're saying, well, this is too hard for me. So I'm soliciting here for people to tell me that they've actually looked at it and they think it's okay. If they have time.

In order to finish my book in 20 years, I'm going to have to be able to write a page a day, and I haven't been able to reach that rate. I'm only getting about 60% of a page in a day. So I have to lower the standard of quality, to do less original work on the pages that I'm writing. But these pages I regard as a fundamental section, so I've put more time into it; I've got original material in there that hasn't appeared in any other publication, and I'd like to have somebody vet it, and check it out.

[Q]: Are you surprised that TEX is still used, that no new system has come along that has surpassed the quality of TEX?

DEK: Actually, there are systems that surpass the quality of TEX but they still haven't apparently taken over. Now the one that's most ... There's the ε-TEX, which has greatly improved features for things like combining right-to-left typesetting with left-to-right typesetting. So I imagine people who are doing typesetting in Hebrew or Arabic are making use of these improvements. And there's Omega, which is Unicode-based and it's a work in progress. There's pdfTEX, I think a lot of you know. But those are still 100% compatible with TEX if you don't use the extra features. I can understand why it's difficult for any other system to displace it, because a lot of the things that have been improved

over TEX really don't matter that much to other people. So TEX only goes 99% of the way. Still, that 1% is like noise, as far as ... So why should I switch to another system if I only have to work a little harder 1% of the time?

I'm totally surprised that TEX has been used as much as it has in so many different parts of the world. I mean, I downloaded a paper two weeks ago from a new journal in mathematics called the *Moscow Mathematical Journal*. I got this paper, and I printed it out, and I have to admit really feeling a thrill that it looked so good. [laughter] You know, here were people halfway around the world and they were using this system that I had done years ago, and what had come out was something that was aesthetically pleasing to me as well as the mathematics — it was what I wanted to read. And there are fifty chapters of TEX user groups around so many parts of the world. And I'm getting newsletters that are published, you know, in Greek, in Indian languages, it's a thrill, and that's a big surprise to me. But I'm not surprised that it's been harder to improve on TEX, because I know how much work there was to get it going.

Excuse me, I haven't been paying attention to people in this part of the world ... [goes to another part of the audience to ask for a question]

[Q]: What do you think is the single most important problem that is left unsolved in computer science?

DEK: Well, computer science has so many subfields that it's really hard to ... Now, if we want to compare it to what Abel did, however, since that's timely, well, Abel solved, you know, the major open problem of his time: Can you express the solution to every polynomial equation of fifth degree with plus, minus, times, and taking roots? And he showed no, you can't. And it's very hard to prove that something is *im*possible. After he worked on that, Abel worked on Fermat's Last Theorem, he worked on $x^n + y^n \neq z^n$ when $n > 2$, and he proved that if x and y and z do exist, they are so huge you could never write them down. He worked on that.

So the question is, if you wanted to be Abel now, and solve the most important problem, it has to be the question about whether P = NP or not. And that is whether or not all the things that we can do with a polynomial number of guesses can be done in polynomial time without guessing. Now, I know only one person who is really working on that seriously at the moment and has a chance of succeeding. In my opinion, that's Andy Yao, who's a professor at Princeton, and inspired by another famous Princeton professor who solved Fermat's Last Theorem. And Andy might very well get it in the next five years. There's a man in Russia who sent me e-mail saying that he has a proof, but, as I say, my life is too short to check everything that comes in.

There was a proof in a Chinese journal a couple of years ago that P = NP, and at first I couldn't find a mistake in it. I worked on it for three hours with some students until we found a bug, and I wrote to the author; still, he didn't believe that I'd found any bug.

But the thing is, that experience made me realize that the problem is academic; it's not going to really be practical. He had given an algorithm to solve the clique problem: If you're given a graph and certain pairs of vertices are adjacent, others aren't, you want to find the largest set of vertices that are mutually adjacent to each other. And he had an algorithm that supposedly solved the clique problem in n^{12} steps. And it was very hard to test that algorithm out because, even if $n = 100$ that's 100^{12} and that's way bigger than I could ... And in order to send him a counterexample, to show that he hadn't got it right, I had to send him an example that we couldn't really run on a computer. Still, n^{12}, that's a polynomial of terribly low degree.

Now, it might very well be that the following scenario happens. Somebody proves ... Actually, there was a big questionnaire sent out to all members of SIGACT, the Special Interest Group on Theory in the ACM, and saying, what did people think is going to happen, you know, how long before somebody resolves this problem? And so on. And I encourage you to read it, it was in the current issue of *SIGACT News*, or the second-last issue.[17] And the opinions went all over the map. But my opinion — which was shared by one other person independently — was that probably in the next fifty years, somebody will prove that P = NP, because there are only finitely many obstructions that would keep it from being unequal. So that means that there exists a polynomial time way to solve everything, but that we don't know what the polynomial is. We might just know that there's some exponent so that everything can be solved in N to that exponent time. However, we may never know what the algorithm is, or the exponent. All we know is that P = NP. Now, that would be the worst possible solution to the problem, because it could never,

[17] William I. Gasarch, "The P = ?NP poll", *SIGACT News* **33**, 2 (June 2002), 34–47.

ever have any use at all, it just would mean that the question was the wrong question to ask.

People make mistakes all the time of misunderstanding the connection between finite things and infinite things. And even computer scientists have trouble understanding that things don't always approach infinity the same way. And when we make judgments about something that's true in the limit, it might have absolutely no connection with our lifetimes.

In the back ...

[Q]: As advice to the young people here, if you were given a second chance and you were starting out fresh in computer science as a graduate student, with all your present experience, what would you do, which things would you go for, and how would you organize your life to achieve that? [laughter]

DEK: That's a great question.

Suppose I'm starting all over completely, how would I reorganize my life? One of the things Abel said, and I guess he got inspired by Lagrange, he said, "Read the masters." He said, in order to learn stuff, don't try to look at everything by yourself, but do look at what other people have done, written in their own words, through the years. But follow your own inclination for sure.

Now, Abel made a bit of a mistake: He didn't take care of having a job, and there is a problem. I was very lucky that I was riding on the crest of waves, so I could get scholarships to go ... I mean, I came from a family where nobody had gotten an advanced degree in college, but I passed exams okay in high school so I was able to get a scholarship to go to a university. Then at the university I could get into computers — computers were just new. So, exciting thing, I found out that I was in this 2% of people who had the ability to think about computers, so I could get jobs, I could get $200 for converting a computer program. Then in the middle of graduate school I was offered to drop out of college and write compilers for a living at an annual salary of $100,000 in 1962 dollars. That would correspond to about $10,000,000 a year now or something like that, not quite as much as a CEO — but anyway I was offered, leave graduate school and get rich, and I decided, you know, I really wanted to spend my life not maximizing the total amount of money that I got. I wanted to do things that were interesting to me, that I thought would be useful to others. And I found that money was a threshold function. If you don't have enough of it, you need it; but after that point, then it's just a problem because you really have to enjoy having money and figuring out what to do with it responsibly, which is something that I never wanted to do.

So my basic idea is to say, whoever you are, you've got a unique combination of talents that's been given to you. Don't decide ... Your life is kind of like a binary search, you try things and find out you did well in this, you try other things, you find out you didn't do so well in that, you go on and continue discovering what are the best ways to use the abilities that you were born with instead of what you think they ought to have been. And you also read what other people have done, and try to learn from that and extend abilities that you have in ways that appeal to you. And then, ... My experience was, once things start to click, then you can start producing and other people will be able to make use of what you did, and you can help them along in the same way. It sounds idealistic, but that's really been the pattern all the way through my life, and it was based on this sort of idea, saying, well, just learn what you're good at, and what you can do that other people might be able to use, that's fun for you.

[Q]: Why can't I type floating point numbers in my `MMIX` assembly code?

DEK: The reason was, it was too hard for me to ..., although I do have the subroutine in there that reads them, I wanted to keep the assembler simple. I didn't see any reason to ... Basically, I was over my page budget; I didn't have that much room to explain what the rules were, so I decided not ... I really believe that an assembler should be stripped down, and that it should be considered as a ... Certainly the second system syndrome could affect assemblers, and I didn't want to get caught with that.

[Q]: Do you see the concept of design patterns as important for programming in the future?

DEK: Design patterns ... I think of all these things as ... In my own experience I don't follow them religiously, but I follow them as motivating ideas for things that I do without using the official tools. I react negatively to somebody who presents me with a tool that's supposed to solve all of my problems, because there are too many people doing these tools, and each one is ... I mean, Linux is a great system, but it's also Unix. Another definition of Unix is "200 definitions of regular expressions sitting in one box". Every part of it is slightly different from every other, and I remember when I first got the manuals for XView and Motif, and you know there were all these things in there; and my reaction was, they call these open systems because the only way to use them is to have twelve books open simultaneously. [laughter]

So the mistake that I see people making is that they say, oh, I'm able to find some pattern, and then I provide hooks so people can plug in, and then they'll be able to use this and it'll do everything. And that, I find, is much less useful than it seems. It's like the analogy I made before, where you place too much reliance on a mathematical theorem, you're not supposed to have to know anything about the proof. My experience is the opposite, my experience is that it is a great surprise to me when I come across a mathematical problem that exactly fits the hypothesis of a mathematical theorem. Usually, it matches almost perfectly to something that I find in a book, but then I have to see how to customize it to my problem. And this ... When I see reusable code, it should be reusable with a little bit of editing, is the way it seems to work for me. I realize my opinion is heretical, but I just have to tell you the way it has been in my own experience.

The architect in Berkeley who came up with the inspiration for design patterns,[18] I've read a couple of books of his, and I think they're quite inspiring. But that doesn't mean I just adopt everything he says. I think everybody adapts what they learn to the things that they were destined to do in their life.

[Q]: I see that you have problems, in your texts, with level 50. How many of these problems have you solved.

DEK: In *The Art of Computer Programming* I have some problems listed at level 50 — these are research problems. How many have been solved? Well, actually, the only one I can remember is Fermat's Last Theorem, so I've replaced it by another one, which is $w^n + x^n + y^n = z^n$, for $n \geq 5$. So I think that will last for awhile.

But anything more than 45 is an unsolved problem. And some of the 46's and 47's have gone down. And one I just changed two weeks ago; the man in Poland, Marcin Petkovic[19] — I can't remember his name for sure — but he ... The question was, the smallest number of comparisons needed to sort 13 elements, and I had conjectured that it could be done in 33 comparisons. No way was known to do it in better than 34; he proved 34 is the best. See, it's kind of a scandal. If you look at optimum sorting, the very fewest comparisons in a comparison-based method of sorting, there's a bound from information theory that says the number of comparisons you need is at least the binary logarithm of $n!$, because if you make binary decisions and you have 13! possibilities, then you have to make $\log 13!$ comparisons, and that was 33. And the best-known way to do even 16 elements is the following: Pick 10 of them; sort those 10 elements by a way that takes $\log 10!$ steps, and that way is known. Then take the other 6, and one by one, insert them among the 10 by a binary search. So that's four more comparisons for each one. And I didn't believe that by the time you got up to 13 you wouldn't be able to think of something better than to put in numbers 11, 12 and 13 each with four. And I still don't think you have to go all the way to 16 this way, but nobody knows a better way to sort 16 elements than to start with 10 and to plug in the other six one by one. But that was a research problem that recently got upgraded. And this guy was a student, and he won the best student award paper in the ESA — what is it called? — the European Symposium on Algorithms; I think it's being held this week or next week.

Another question is, when do you folks get to go home? [laughter]

[Q]: I'm a person that doesn't have an operating system question to ask. What do you think of the quality of today's programming languages?

DEK: The quality of today's programming languages ...

Well, they always seem to be ... All programming languages have this nth-system syndrome, where they take what they know how to do well, and they clean it up, and then they add a new feature, that they *don't* understand. [laughter] And all the time this has been true. Nobody has ever said, no, I'm not going to add any new features in my new language, I'm just going to do everything better. I mean, like if you take the subset of Ada that corresponds to Pascal, you have a beautiful language compared to Pascal. And there's a big problem with Java being inherently unstable, because the library, if I understand it correctly, it's impossible for a user program to create a subroutine that isn't in the main Java library, for which the people to whom you want to distribute this program can use exactly the same syntax they would use if it were in the standard library. They have to use a different syntax for your system than when they're using a real certified subroutine. And so every time somebody finds that there's a gap in the standard library, they have to add to the standard library, and every year the standard library is going to have to get bigger and bigger, unless they ...

I worked hard in METAFONT so that this would never happen, that somebody could add an extension to METAFONT in such a way that a primitive

[18] Christopher Alexander, *A Pattern Language* (1979)

[19] Marcin Peczarski, *Lecture Notes in Computer Science* **2461** (2002), 785-794.

would look as if it had been built in. Apparently the designers of Java did not do this. But I'm not an authority on that because I never read all the details of Java. My experience is that there's never been any stability in the programming language I need to use, so I started writing programs in C−−, which is the subset of C that I like. [laughter] And then I discipline myself to keep information private that's supposed to be private, and things like this. But I'm not using ... All of today's languages suffer from some serious problems, and I don't see that the situation will stabilize unless developers come to a situation where they don't want to do something new, they just want to take the things that they really understand ...

With TEX it was a different ... Why did TEX become stable? The reason was kind of stupid, but because I *didn't* want TEX to do everything. You know, I wanted most of all to find the cheapest way, to implement the least I could possibly do, so that I could get out of it without being totally irresponsible, and go back to writing *The Art of Computer Programming*. But quite seriously, I didn't want to spend my life coming out with something better and better and better for typesetting. I wanted to get something that would go only to a certain level, and I spent five years on the endgame, always asking "How am I going to stop, to keep it so that I don't have to do it again?"

I thought I had found a stopping point, and then I learned about Europe. No, not really. [laughter] But I hadn't made it easy to do the Norwegian alphabet, and other things that needed 8-bit codes. I had assumed that nobody would ever design keyboards where it was easy to enter 8-bit codes. And as soon as it became easy to enter such codes, there was pressure from all over the world saying, well, let's be able to do that with TEX, so I had to spend another six months making TEX as if it had been designed for 8-bit codes in the beginning. That was a big ... But my attitude was never to wake up in the morning and say, well, how can I improve TEX to make it typeset better? No, my attitude in the morning was, how can I finish this and get it off my back? [laughter]

That's the only recipe that I have for having a system that's going to become stable.

Tom Lyche: Well, I think we're ready for this to come to an end. I'd like to thank you very much for this lecture, and I won't ask if there are any questions. [**DEK** laughs, along with the audience]

Professor Knuth will also be here on Tuesday, where I will remind you that he is giving a lecture, a scientific lecture here on Tuesday. I hope many of you will attend that. So I think we give him a big round of applause.

[extended applause]

Narve Trædal:[20] Here is a little symbolic gift from the University: two small wine glasses, with the University logo on them. [**DEK**: Oh, great!] Thank you for your support today and through many years. We are deeply honored and grateful. Thank you so much. [applause]

[20] head of department administration