Interview: Donald E. Knuth

Posted 25 Jan 2000 by advogato

This week, Advogato had the pleasure and honor of interviewing Prof. Donald E. Knuth. He is the author of the TeX typesetting system as well as The Art of Computer Programming and a number of deep, insightful papers and books. The interview took place by phone on a rainy California winter day. The topics covered the freeness of TeX and its fonts, how TeX's innovations have slowly diffused into commercial systems, some history of math typesetting, the design of TeX from the beginning as an archival system, literate programming in the age of the Web, MMIX and the Transmeta chip, how to avoid generating inscrutable error messages, and taking the TeX ideas to a broader community. Read below to find more about a remarkable person.

Advogato: **The first questions that I have are about free software. TeX was one of the first big projects that was released as free software and had a major impact. These days, of course, it's a big deal. But I think when TeX came out it was just something you did, right?**

Prof. Knuth: I saw that the whole business of typesetting was being held back by proprietary interests, and I didn't need any claim to fame. I had already been successful with my books and so I didn't have to stake it all on anything. So it didn't matter to me whether or not whether I got anything financial out of it.

I see.

There were people who saw that there was a need for such software, but each one thought that they were going to lock everyone into their system. And pretty much there would be no progress. They wouldn't explain to people what they were doing. They would have people using their thing; they couldn't switch to another, and they couldn't get another person to do the typesetting for them. The fonts would be only available for one, and so on.

But I was thinking about FORTRAN actually, the situation in programming in the '50s, when IBM didn't make FORTRAN an IBM-only thing. So it became a lingua franca. It was implemented on all different machines. And I figured this was such a new subject that whatever I came up with probably wouldn't be the best possible solution. It would be more like FORTRAN, which was the first fairly good solution [chuckle]. But it would be better if it was available to everybody than if there were all kinds of things that people were keeping only on one machine.

So that was part of the thinking. But partly that if I hadn't already been successful with my books, and this was my big thing, I probably would not have said, "well, let's give it away." But since I was doing it really for the love it and I didn't have a stake in it where I needed it, I was much more concerned with the idea that it should be usable by everybody. It's partly also that I come out of traditional mathematics where we prove things, but we don't charge people for using what we prove.

So this idea of getting paid for something over and over again, well, in books that seems to happen. You write a book and then the more copies you sell the more you get, even though you only have to write the book once. And software was a little bit like that.

I think that's the model that software publishing generally comes from. There was a quote that you had in the "Mathematical Typography" essay reprinted in "Digital Typography" where you said, "Mathematics belongs to God."

Yes. When you have something expressed mathematically, I don't see how you can claim... In the context, that was about fonts. That was when I had defined the shape of the letter in terms of numbers. And once I've done that, I don't know how you're going to keep those numbers a secret...

Proprietary.

I can conceieve of a number that would be a million digits long and would be extremely expensive to compute, and once somebody knew that number, it would solve all kinds of problems. And I suppose that would make it a little bit harder to say that God already had given us this number, when it's a number that you can only discover by a tremendous amount of sweat.

When I made that quote, I didn't think of such things.

Fonts seem like a really interesting edge case for that argument, because a font is in some ways a mathematical formula, especially a TeX font, much more so than what came before, but it's also an artwork.

Absolutely. It absolutely requires great artistry. So the other part of this is that artists are traditionally not paid like scientists. Scientists are supported by the National Science Foundation to discover science, which benefits the human race. Artists, or font designers, are not supported by the National Font Foundation to develop fonts that are going to be beneficial to the human race. Fonts are

beneficial to the human race, they just don't traditionally get supported that way. I don't know why. They're both important aspects of our life. It's just that one part has traditionally gotten funded by a royalty type mechanism and the other by public welfare grants for the whole country.

Perhaps that has something to do with the absolute necessity in science to have open access to the results of others, that if you did science in a closed, proprietary framework that the disadvantages would be so clear.

With fonts, it was pretty clear to me.

Ok! That's a question that Federico Mena Quintero suggested. You've gotten a number of free fonts contributed by artists, in some cases very beautiful fonts, to TeX and to the Metafont project. In general, this has been a real struggle for open source development these days, to get free fonts. Do have any thoughts?

I think it's still part of this idea of how are the font designers going to get compensated for what they do. If they were like a scientist, then they've got their salary for doing their science. But as font designers, where do they get their salary? And musicians. It's just a matter of tradition as to how these people are getting paid.

But how did you address those problems with the fonts that got contributed to TeX?

In my case, I hired research associates and they put their fonts out into the open. Or else, other people learned it and they did it for the love of it. Some of the excellent fonts came about because they were for Armenian and Ethiopian and so on, where there wasn't that much money. It was either them taking time and making the fonts or else their favorite language would be forever backwards, so I made tools by which they could do this. But in every case, the people who did it weren't relying on this for their income.

If we had somebody who would commission fonts and pay the font designer, the font designer wouldn't be upset at all about having it open, as long as the font designer gets some support.

And you did some of that.

Yeah. In fact, I worked with some of the absolute best type designers, and they were thrilled by the idea that they could tell what they knew to students and have it

published and everything. They weren't interested in closed stuff. They're interested in controlling the quality, that somebody isn't going to spoil it, but we could assure them of that.

Right. Working with the creator of the software.

Yeah, if they didn't like the software, I could fix it for them.

One of the things that struck me when I was reading "Digital Typography" is the intensive study that you did, especially in the area of math typesetting. When I was writing papers, using math formulas in TeX, I just typed in the commands and out came the math and it looked pretty good to me. It shouldn't have been surprising, but it definitely struck me how much attention you paid to the best mathematics typesetting of past centuries.

I do strongly think that people, when they start throwing computers at something, they think that it's a whole new ballgame, so why should they study the past. I think that is a terrible mistake. But also, I love to read historical source materials, so I couldn't resist. I had a good excuse to study these things, and the more I looked at it, the more interesting it was. But I don't think responsible computer scientists should be unaware of hundreds of years of history that went before us. So that was just a natural thing to approach it that way, for me.

I noticed, for example, that in the proprietary software market for publishing, that systems are only today acquiring features that have existed in TeX for a long time, for example whole-paragraph optimization. There's a big to-do about Adobe InDesign, which finally...

They finally implemented the TeX algorithm.

Did they implement the TeX algorithm?

Yeah, that's what they said.

Did you talk to the people?

I met three of four of them at the ATYPI meeting in Boston in October, but that was after I had heard about it, that some friends had found this in the documentation.

Another similar issue is TrueType fonts. TrueType fonts have this property

of including instructions, computer programs effectively, in the font, to do hinting.

Well, I never met Elias or whatever.

Sampo Kaasila?

I don't know. I know enough about TrueType to know that it's a very intelligent design, that is similar to Metafont except that it strips out everything that's slow. So the way the hinting is done is by program, certainly. Of course, it came out maybe ten years after Metafont, so probably something got through somehow.

There was the F3 font that Folio was making, if I can remember the name, what the people in industry called it. Some of the people that I had worked with on Metafont went into making font designs that were similar to TrueType, but have not been successful.

There's a fairly major controversy with TrueType right now, that there a number of patents that are owned now by Apple. It's kind of interesting to me that that is the case even though it's for the most part derivative work of what was in Metafont.

I've been very unhappy with the way patents are handled. But the more I look at it, the more I decide that it's a waste of time. I mean, my life is too short to fight with that, so I've just been staying away. But I know that the ideas for rendering... The main thing is that TrueType uses only quadratic splines, and that Type1 fonts use cubic splines, which allow you to get by with a lot fewer points where you have to specify things.

The quadratic has the great advantage that there's a real cheap way to render them. You can make hardware to draw a quadratic spline lickety-split. It's all Greek mathematics, the conic sections. You can describe a quadratic spline by a quadratic equation (x, y) so that the value of f(x, y) is positive on one side of the curve and negative on the other side. And then you can just follow along pixel by pixel, and when x changes by one and y changes by one, you can see which way to move to draw the curve in the optimal way. And the mathematics is really simple for a quadratic. The corresponding thing for a cubic is six times as complicated, and it has extra very strange effects in it because cubic curves can have cusps in them that are hidden. They can have places where the function will be plus on both sides of the cubic, instead of plus on one side and minus on the other.

The algorithm that's like the quadratic one, but for cubics, turns out that you can

be in something that looks like a very innocuous curve, but mathematically you're passing a singular point. That's sort of like a dividing by zero even though it doesn't look like there's any reason to do so. The bottom line is that the quadratic curves that TrueType uses allow extremely fast hardware implementations, in parallel.

The question is whether that matters of course, now that CPU's are a zillion times faster.

But for rendering, Metafont was very very slow by comparison, although I'm amazed at how fast it goes now. Still, it has to be an order of magnitude better, and certainly that was a factor in getting TrueType adopted at the time that it was, because machines weren't that fast then. So TrueType was an intelligently chosen subset, but certainly all the ideas I've ever heard of about TrueType were, I believe, well known in the early '60s.

Back to this issue of preserving the past. I was reading some papers of Edsger Dijkstra. For a while, he used handwritten manuscripts and then a typewriter to actually distribute the work. And, his notation became much more typewriter-like, that he would use an underlined A or a boldfaced A instead of the traditional \forall symbol.

I've gotten some of his handwritten notes, but I don't remember the typewritten ones.

I was looking at the proceedings of the Marktoberdorf summer school in '90, where there were a couple of papers by him and his group. In any case, it occurred to me that TeX has made the traditional mathematical notations so accessible to practicing computer scientists, students, researchers, etc. It's very likely that if there hadn't been something like TeX, in other words if mathematical typesetting had remained strictly in the domain of book publishers, and people who did publishing as their profession, it's likely that the standard notations in computer science would have become much more typewriter-like, kind of ASCII-ized.

That's interesting.

As it is, if you look at Principles of Programming Languages, there's all this type theory in there with beautiful Greek letters, inference rules and so on. The visual aesthetic seems to have been inspired by the easy availability that TeX provides.

Certainly, TeX was partly influenced by the needs of computer science, that pushed beyond what mathematicians had needed. Computer scientists needed to see mathematical structure, but they also needed to see the kind of structure you have in programs. So we had to push the envelope in the ACM Journal in the '60s. To publish computer science papers, the typesetters using hand methods in the '60s were being forced to work harder by the computer scientists than by the mathematicians at that time. It's part of the need for a way to see the structure that's in the stuff you're working with. Since I'm writing books about computers, naturally I made TeX so that it could do that.

It's clear that TeX was inspired by the visual aesthetic of mathematics.

Oh yeah.

I'm just saying: to what extent mathematics and computer science today is being influenced by the visual aesthetic of TeX?

Well, I don't know. I think the fact that TeX was open-ended means that people can choose their notation themselves. They don't have to adopt somebody else's macros particularly. That was the bind that we were in before. We would have to first write our paper, then explain it to the printer, and the printer would maybe get it right. But now we're less hesitant to use a notation that's not traditional, but that we think is appropriate, because we know that we don't have to go through a noisy channel in the middle.

One of the other accomplishments of TeX that I continue to be impressed with is the consistent rendering, the idea that you have a source file and that on virtually any implementation of TeX, you'll get the same results.

That's what I insisted on the most. I didn't want to get paid, but I didn't want it to change.

If you look at, for exmaple, a lot of the new systems that are being designed today like Cascading Style Sheets for the Web, they allow a tremendous amount of latitude for the implementor, so you don't get anything approaching consistent rendering.

With the Web, it's a different tradeoff, between appearance and the ability for a search engine to find it.

Presentation vs semantic markup, right, in general?

Right.

But it seems to me that TeX's model is actually a way to be better in both of those regards than the way that the Web has been evolving, because you do have at least the potential for semantic markup, and then you get the fact that you know what the presentation will be. So, one example from the Digital Typography book is that you described changing the wording of some of the exercises so that the symbols would line up better, that the parentheses wouldn't collide with descenders.

This wouldn't go too well to the web, because people can certainly set the width of their browser window differently, and I'm not expecting that I would optimize for all of those things. But if I'm optimizing for a particular page size, then I would say, "why not." Mostly, the rewriting is that I can get a figure to be on the same page where you want to see it, without turning the pages over, if you care about things like that. There were some parts of The Art of Computer Programming where I said, well, I gotta think of five lines of things to say on this page, and then I would think of ten lines. It was a motivation for research. I knew I needed five more lines of stuff to say on this page, because otherwise the illustrations aren't going to come out right.

If you are going to print, and of course a lot of people print from the Web, and usually on page sizes that are well defined in advance, with TeX you have at least the possibility of doing that. If you do reformat the width, then the semantic markup is still there so maybe it isn't as optimized, but you still get a reasonable rendering.

Well, what I was mostly concerned about was that the rendering would be the same in ten years. What you say is probably important, but I didn't have that so much in mind as the fact that on everybody's machine it would come out the same, not only now but in the future. Because there was so much software rot going on where stuff wouldn't work any more. And I didn't have time to keep updating... It's going to take many years to finish TAoCP, so I wanted to make sure that I could make Vol. 7 match Vol. 1.

But that seemed to be at odds with something else you said in "Digital Typography" about being surprised that people didn't make more custom versions of TeX for specialized applications. And if that had happened, and if some of those had become popular, wouldn't that have created similar problems?

I wasn't expecting them to become popular. I was expecting that they were going

to be one-off things. If I were a publisher, that's what I would have done.

But if you create a one-off, that still creates the potential for software rot, that ten years from now...

Okay. I have to save my program. Yeah.

And that's hard. Usually.

The program is there in a bunch of change files, so as long as the C language isn't changing, then I'm ok.

Well, it isn't changing much.

See, the way I would make these custom version is by using the change file facility that we built in when we started distributing TeX. It means that you have a master file, and then you have another file that says, these are the things I changed. And that's the way people ported TeX to lots of different operating systems, they would have it in these change files. I could change the master file, and their change file would probably still work.

I see. I didn't actually know about that aspect of the TeX distribution.

There's a change-file mechanism that's been quite successfully received in all the TeX distributions.

That sounds like it could be useful in other projects as well.

Isn't hasn't been picked up yet, but I don't know why [laugh].

Speaking of literate programming, the question that I have for you is: now that people are moving everything to the Web, including programming, do you think that's another chance for literate programming to become popular?

There's a lot of ferment in this direction. I still don't have the dvipdf program that I got to get installed on my Linux. I guess I gotta try it. But a guy has worked out now that he can convert to Acrobat format, a literate program automatically come out in Acrobat format, so that you can use all the features of the Acrobat reader to click on stuff to move around in the documentation.

Cross-referencing and so on?

Yeah, find a variable where it's declared, and all other uses of the variable. It's certainly a natural thing for hypertext, and this guy in Brazil has worked out a nice system. Still, the thing that's holding it back is probably that some programmers just don't like to document their stuff.

That's certainly a problem we've had in free software, that if there's a documentation file and then a code file, that the two often get out of sync.

These are slowly changing, but most of the horror stories that you hear about that stuff is because of the illiterate nature of the program.

So you think that literate programming is a way to make that better?

It's so much better than the alternative, but I think Jon Bentley explained it to me best, not much percentage of the world's population is really good at programming, and not much percentage of the world's population is really good at documenting, and here you need to be doing both. [laughs]

So, I think that in my experiments with Stanford students I found that more than half of the students that I worked with really hit it off well with literate programming. Maybe Stanford students aren't average. [laugh]

Right.

So you need somebody who's not afraid to write an essay, as well as not afraid to write a computer program.

They work together perfectly, but you have to be able to communicate to the computer, and you have to be able to communicate to the human being, and if you don't do both, then you can't expect your program to be as successful. Literate programming is just the best way I know to do both at the same time.

My new book on MMIX, where another example of what I hope people will consider is just a state-of-the-art, normal way to document programs. In this case, the program that I wrote for this MMIX simulator last year, I wouldn't have been able to finish the program without literate programming. It was just too mind boggling. Not only was literate programming a nice add-on, without that tool, I wouldn't have gotten the program done.

I saw the MMIX book just came out. I haven't gotten my copy yet, but I'm

really looking forward to it. That is the new instruction set that you're using in TAoCP, right?

Yeah. Well, I'm going to use it eventually. It's there, but it's going to be a while before I replace it in volumes 1, 2 and 3. I gotta write volume 4 first.

MMIX is another digression for you, then.

It'll be used a small amount in Vol. 4. If I talk about machine code at all in Vol. 4, there are a few places, then it will be in there.

I was reading up on MMIX at the same time that the Transmeta processor came out, the Crusoe. I don't know if you're familiar with that, but it's a VLIW that does on-the-fly translation of a pretty much arbitrary instruction set into its own native format. Do you think that MMIX is really going to be very similar to the architectures that people will need to know, or is the increasing complexity of CISC chips and new things like VLIW going to make...

I don't know. These VLIW chips like the IA64 are getting very mixed reviews. I guess the Pentium translates into a RISC-like language internally.

Yeah, sort of.

I think in any case, this is a model that is going to be as close to efficient and clean at the same time as we're going to get in the forseeable future. I wanted it to be realistic, but I also wanted it to be clean and easy to learn.

One of the things that intrigued me about this Crusoe chip that Transmeta is doing is that they have firmware that does dynamic translation of the source architecture. Obviously they're pushing x86 as the most commercially important source language, but I was just wondering if maybe MMIX would be a good candidate for...

If they want to play with it. But that will be just an academic exercise right now rather than a commercial one, because there isn't that much code written for it.

Well, the x86 has some serious problems, that the memory bandwidth is a lot higher than it needs to be because it doesn't have enough registers.

Oh, absolutely.

So, for this particular chip, it might make more sense for gcc to generate MMIX code.

Oh, I see what you mean. Well, that's interesting.

I don't know. I was just wondering if you had some thoughts on that.

Well, okay. So we need a library, all that kind of code. It's for the MMIXmasters to develop, because I haven't got time to mix with it.

The other thread I was interested about was this whole development of commercial TeX packages in the '80s, for PC's and so on. Those didn't catch on, and PostScript did, and the world is quite different because of that. Do you have have some stories about why that would have happened, or what you might have done differently?

Those programs were kind of heroic. At that time, PC's were limited. If you wanted to have an array with more than than 65kbytes in it, you had to go in to a special extended addressing mode. So it wasn't easy to port TeX to a PC in those days, with the 186 and the zero-86.

Except in the mathematics and physics community, there weren't that many people who were interested in the archival quality of stuff, or making something that looks best instead of good. I never expected TeX to be the universal thing that people would turn to for the quick-and-dirty stuff. I always thought of it as something that you turned to if you cared enough to send the very best.

It made me almost scared when I heard that college students in Japan were being forced to learn TeX, because I never wanted anybody to be compelled to dot their i's and cross their t's. I always thought of it for somebody who wanted something to pride in putting a little spit and polish on documenting. I didn't think of it as the world standard or something like that at all.

The things that, in my experience, are the most difficult for people who are learning TeX or using TeX, are for one thing, the difficulty of using PostScript fonts. And from what I understand, that's accidental.

Oh, there's no problem whatsoever, except since on every machine you needed to interface a different way, and everybody had a different subset of the fonts. There were six different PostScript Times Roman fonts, and they all had slightly different spacing, so there was no way to make that easy. Then Adobe was going to put out another Times Roman, because they could improve it. There was no idea of

archiving these fonts or keeping them in a way that wasn't going to change from year to year the way other things in TeX were doing, so that was one of the hangups. Also, we didn't have that many drivers until about 1990 for PostScript. I had been using them for different projects, but since I wanted to make sure that I was technology independent for my own books, I did everything in bitmaps that I generated in a form that I knew would work 50 years from now. But when I would do a special project that was not for my books, but for something where I was supposed to make camera ready copy for some journal, then I would use commercial fonts. We finally made the virtual font mechanism popular, and people started understanding it and it took another ten years. Now, that's being used a lot.

It's better, but it's still difficult to just take a PostScript font and drop it into TeX and have TeX use it. That might be more of a packaging issue.

Yeah, You've got to change a couple of lines in some mysterious files that are not well documented.

And I've had problems with encodings too.

You have to be sure that you get the right font metrics.

It just seems like a hassle when I've tried to do it.

Well, it's a hassle to install new fonts right now, because the TeX world and the Microsoft world are separate.

Yeah.

So you've got to install the fonts in a few different places.

The other issue that is definitely intimidating for beginning users are the error messages and error reporting, especially when you're using a macro package like LaTeX. It seems like the error messages that you get have virtually no relationship whatsoever to what you did wrong. I'm wondering, is that because of the macro nature of that process, an inherent limitation of systems that do what TeX does by evaluating huge piles of macros?

There is something inherent. If you have a language that has a certain global character to it so that if you make a mistake on line 100, there's no way for the computer to find out about it until line 200, because even though you made a mistake, it still made sense. Then you get into the situation that you're describing, where error messages are pretty inscrutable.

So if I had put more care into making a language that was more restrictive, that it would check a lot more stuff, then you'd catch an error much quicker.

Right.

Let me explain. TeX does have some things like this. It'll say that a macro can be defined to be \outer, which means that you're not supposed to have this macro in the middle of a parameter to another macro. So if somebody forgot a right brace, then it'll catch it before you've gone too far in the program. But suppose that I had made something that you have to end something on the same line it begins. For example, take a programming language. If you have a string starting with a double quote mark, you've got to finish the string on that same line. So if you forget a double quote mark, you know about it right away. But if you didn't have that rule, then you could leave out one double quote mark, and everthing that was outside of a string would be treated as inside of a string, and vice versa.

And you generally get an error message the next time a quote mark appears in the program.

Yeah, you're bound to get an inscrutable error message. So, there's part of it inherent that TeX is a language that allows you to go on and on and still be syntactly correct.

And the other problem that I see with macros is that it makes it much more difficult to do an incremental update to a display. Of course, that's something that seems very interesting now, with big high-res displays commonly available, and probably less important when TeX was originally being created. If you make even a small change to a TeX document, you pretty much are forced to go through...

TeX doesn't know that your small change hasn't changed everything.

Right. So you can't really have a version of TeX that lets you edit in one window and have those...

But on this Pentium III, TeX just generates 50 pages by the time you hit carriage return.

Yeah, but there's still a difference between that and incremental editing where you're actually typing and seeing your changes updated in real time.

It's so close to real time now, even if I'm going through dvips and Ghostview, and going up to page 20. Right now, my Linux machine at school is ten times faster than my SparcStation at home, so it's worthwhile for me to go to school, just to get this effect, when I'm doing a lot of editing of spacing. So I'm sitting here with a 50 page document, and I say, "maybe I'll add a thin space here," and I'll re-TeX the entire document, and I'll re-convert to PostScript, and it's just a fraction of a second. And I can see right away that the thin space has gone in there. I can't believe it. And this is going to be so much more common in the future.

It's hard for me to believe because I know what TeX is doing, I know what dvips is doing, and yet it's fast, it's done.

Well, you've heard of Gates' law, which is that the speed of software halves every 18 months, right? [laugh] And I guess that TeX is an exemption to that.

Well, it hasn't changed.

Exactly. Well, one of the things that I'm personally interested in is making the kind of stuff that's in TeX more accessible, to users in general, so I'm working on interactive editing in the Gnome project, which is a Linux thing. I'm very interested in doing things like the whole paragraph optimization and the nice hyphenation with all the different languages. Also, in the area of graphics, I've been doing work with splines and using some of the work that John Hobby did for the smooth fitting of splines to just the control points, so you don't have to fiddle with the Bezier control points. I've got some prototype software around that does that interactively. I'm very excited by it, because I feel like somebody who is not a mathematician but who is a font designer might be able to sit down with that, and really get the benefit of a lot of the ideas that were in Metafont, but not have to deal with it as mathematics. You just deal with it directly on the screen. Now, to you, you're a mathematician.

No, no. That's right. It never became natural. I tried to explain how, but even the idea of multiple masters was a pretty hard sell.

Yeah. But in any case, I'm very grateful to have the enormous body of work that's been done by you and your graduate students on TeX, Metafont, and related packages. My work is so much easier. All I have to do is adapt this stuff and steal all your best ideas.

Well, it's not... The hard stuff is still getting it so that it responds right to the people who need it. For that, you have to really be in that community. Also, to see how

the users...

That's what I'm trying to do.

Like this error message business that you brought up. You have the same kind of thing in a C language program, where if you do something wrong, you might get a hundred error messages. But the amazing thing about the literate programming tools, CWEB, is that if I make five CWEB errors, I will tend to get five error messages.

Interesting.

Not only the first one will make sense, but I'll be able to look at the whole thing and fix them all and recompile. And it's always been that way. There's something about that design that localizes the errors. I'm not sure what the key is.

Well, that would be interesting to look into.

That's one of the tradeoffs that you have between how much redundancy is there so that the computer can check as you're going. If you're going to catch an error right away, that means that people have to write things a little bit redundant. That means that you put a little bit of burden, but maybe it's still better to have that little bit of burden than to have your errors become inscrutable.

Aha. Unless you're one of those people that always types things right the first time.

Oh, no no. I have never met such a person.

All right, that's all the questions that I had.

Good luck to you.

It's really been a pleasure.

Dr. Fun, posted 10 Feb 2000 by xach

> I think today's Dr. Fun is somewhat relevant.