Computer Literacy Interview With Donald Knuth

By Dan Doernberg
December 7th, 1993

CLB: You have just-published books on both CWEB and the Stanford GraphBase, two areas of your own research. Let's start with CWEB, which integrates C and TeX to facilitate program documentation.

Knuth: The CWEB system is an add-on to C that makes programming better than any other method known in the world, by far. I simply have to be honest and say that it's the greatest thing that's there. The CWEB System of Structured Documentation is the definitive user manual and complete explanation, more than anybody really needs to know about CWEB.

CLB: You've said that CWEB gives an order of magnitude improvement in programmer productivity--- how so?

Knuth: Well, maybe not an order of magnitude, maybe only a factor of two. People who have used CWEB have noticed that they write better programs, that the programs are more portable, more easily debugged, more easily maintained... and they don't take as long to write.

CLB: Has CWEB been used just at Stanford, or in industry as well?

Knuth: It's being used around the world. We've had WEB, the original version (for Pascal) in a variety of systems, and then more and more people started getting infected by it. TeX was written in WEB. Silvio Levy did the conversion to CWEB in 1987. It was experimental for a long time, and now I'm just saying "The experiment worked!". CWEB is much better than WEB, because C is a much nicer language to work with for system programming and lots of other things. For anybody who really cares about programming, I have no idea why they would not prefer this to any other system.

CLB: Easy to use, runs fast, all that good stuff?

Knuth: Right, and it makes you happy after you finish writing a program!

CLB: Even if you write a bad program?!

Knuth:(Don's wife--Ed.) Almost... well... yeah! Jill will tell you, I come out of my office several times a week saying, "CWEB programming is such fun!" It's true, I just can't do enough of it.

The frame of mind that you're in when you're writing a CWEB program is that much better than the old attitude. You think of yourself as writing for a human being, explaining to a human being what a computer should do, instead of thinking of yourself as talking to the computer telling it what to do. You get your act together better when you're explaining it to another person. This approach helps even for a program that you're going to throw away after an hour. CWEB is a tool that I recommend using even if you're writing a program only for yourself, for your eyes only.

CLB: CWEB seems very close to the structured programming models of the 70s...

Knuth: Right, it's the next step. With structured programming, there were some people saying program top-down, and others saying program bottom-up. With WEB/CWEB you can do parts of it bottom-up and parts of it top-down, whatever you feel is right for the program, or for the part of the program you're in.

The structured programming methodology was great... but the way to really understand it is not as a cookbook of rules, but as a way to understand the relation between high-level and low-level views of a program. The way you do that is by viewing the program as a web, as a bunch of small pieces that are simple in themselves and that have simple connections to other small pieces. This way of understanding the complex whole in terms of simple small parts, and the connections between those parts, is supported by the WEB scheme.

You can create the parts in whatever order is psychologically best for you. Sometimes you can create them from the bottom up. Bottom-up means that you know somehow that you probably need a subroutine that will do something, so you write it now while you're ready, while you're psyched for it. With this bottom- up programming, your pencil gets more powerful every page, because on page nine you've developed more tools that you can use on page ten... your pencil is stronger.

With top-down programming you start at the beginning and say "I'm going to do this first and then this, and then this"... but then you have to spell out what those are--- you can wind up gasping for breath a hundred pages later when you finally figure out how you're actually going to do those things!

Top-down programming tends to look very nice for the first few pages and then it becomes a little hard to keep the threads going. Bottom-up programming also tends to look nice for a while, your pencil is more powerful, but that means you can also do more tricky stuff. If you mix the two in a good psychological way, then it works, even at the end.

(TeX: The Program--Ed.) I did this with TeX, a very large program: 500+ pages of code in the book . Throughout that entire program, all those lines of code, there was always one thing that had to be the next thing I did. I didn't really have much choice; each step was based on what I'd done so far. No methodology would teach me how to write a piece of software like that, if I followed it rigorously. But if I imagined myself explaining the program to a good competent programmer, all that this long program was, then there was just this one natural way to do it. The order in which the code appears in the book is the order in which I wrote it.

CLB: To what extent did you or do you follow the "holy war" debates about software engineering methodologies?

Knuth: I didn't follow every nuance of that work, but I was aware of the dominant ideas. I didn't know what the CASE tools were until many years after other people did. I think it was bad to make too much of a religion out of it. There was a lot of "political correctness" about how to program in those days.

There was a similar thing in the mathematics community in the 1920's, where people were saying that good mathematicians would have to prove theorems a certain way. You weren't supposed to use certain tools of proof that some people thought might lead you into paradoxes. It was like trying to do mathematics with a hand tied behind your back. Similarly, politically correct structured programming was keeping people from getting good programs done, when they knew perfectly well what they were doing, just because their approach didn't happen to fit with the current idea of correctness. Computer science is like every other field; it goes in waves of fashion. Some of the trends are good, but almost every good idea seems to get used in a different way than it should have been.

For example, take random number generators. People had no theory about how to generate random numbers for fifteen years. Then somebody proved one small result about a particular method: if you averaged the serial correlation over an entire period of a billion numbers, the average would be zero, which was good. All of a sudden, everybody switched over, they took out all their old routines and converted to this new method, because it was the only one that had any theory to it whatsoever. It turned out this was a horrible random number generator; the theory had not noticed that the average over the first half was +1 and over the second half was -1! All through history, people have taken ideas and misunderstood the limitations of them.

CLB: Which method was this?

Knuth: Well, it was called RANDU in most subroutine libraries. It's been pretty well purged by now; still, if anybody sees a subroutine named RANDU, get rid of it!

CLB: Did you integrate WEB with C because so many programmers today are using it, or do you personally like C and write with it?

Knuth: I think C has a lot of features that are very important. The way C handles pointers, for example, was a brilliant innovation; it solved a lot of problems that we had before in data structuring and made the programs look good afterwards. C isn't the perfect language, no language is, but I think it has a lot of virtues, and you can avoid the parts you don't like. I do like C as a language, especially because it blends in with the operating system (if you're using UNIX, for example).

All through my life, I've always used the programming language that blended best with the debugging system and operating system that I'm using. If I had a better debugger for language X, and if X went well with the operating system, I would be using that.

An extreme case occurred one year I worked in a lab where the operating system had been designed by Ned Irons. The system was for one of Cray's early machines, and Irons had also written a compiler language called IMP. IMP had a lot of horrible features. One, it was an extensible language, and everybody in the lab would keep extending it. A program that worked on Monday wouldn't work on Tuesday, and the first thing that you'd do if your program failed was to check whether the compiled code was OK. The second thing about IMP was that it was an extremely terse language. For example, where in PASCAL you would say "IF X > 0 THEN...", in IMP you say "X+=>". In other words, your program was very short. You felt like you were writing elegant programs, because there were only a few characters, but you couldn't read them the next day! Being very terse meant that you couldn't fathom this bunch of marks on the page...

CLB: I realize your current emphasis is on "literate programming", but were you ever whatsoever attracted to APL as a math-oriented language?

Knuth: That's another story. APL is for people who have problems to solve and don't care too much about efficiency; they want a nice elegant way to state the solution to their problem, but the solution that they come up with is not necessarily anything that a computer has an easy job doing. It's a problem specification language, but not a system programming language... there is an APL-WEB.

But I want to say more about IMP. The third thing against it was, if you made a mistake, the compiler would either get into an infinite loop, or it would stop on your first error and say "ERROR ERROR ERROR" and quit; you would have to figure out what the mistake was! It was not a great language or compiler.

However... it was still my language of choice, because it fit that operating system perfectly. The arrays would be named in a way that you could easily see in the debugger, and you could know where the storage was being allocated, you knew what was going on, and you could actually get your program running reliably, because IMP blended with the operating system. You couldn't do that with any of the other languages. You might be writing with a better language, but you would get your work done a couple of weeks later, instead of getting answers. I used IMP.

CLB: Was IMP being used at Stanford?

Knuth: It was at a research lab in Princeton. A year before I came to Stanford, I worked there on a classified cryptanalysis research project.

CLB: Please tell us about your other new book, The Stanford GraphBase.

Knuth: The GraphBase book is for two kinds of people. It has a research purpose; the people who are working on the study of new algorithms for combinatorial problems need a standard set of test data on which to compete with each other, and for benchmarks. As I was preparing Volume IV of The Art of Computer Programming, I decided that I would make all the examples and data that I'm using in that book available to everyone. There was a need for some standard benchmarks, and everything should be well arranged so that it is easy to use in thousands of ways. So... I now have a collection of thousands of standard data sets; anyone in Poland can have exactly the same data as anyone in California or China. It's very portable, and can be downloaded from the Internet.

The second purpose of the GraphBase hook is that it is an example of CWEB programming--- it's actually 32 examples of CWEB programming. They're short programs that illustrate the programming style that I prefer. The examples are like little essays, little short stories of computer programs, that are perhaps fun to read.

CLB: What is your current hardware and software environment?

Knuth: I use CWEB for my programming. I use the Emacs editor very heavily, and I use a great high-level language called METAPOST for drawing technical illustrations. This is a new language by John Hobby that is going to be released soon, I think. It's based on METAFONT. 75% of the code is mine from METAFONT, but it's fixed up so that it generates PostScript. I love it.

I also use Mathematica. The people at Maple are trying to convince me to switch over to Maple, another excellent system. At the moment, I like Mathematica because you don't have to type your multiplication signs; you can say "2X" instead of "2*X". Also, the Mathematica manual is exceptionally good.

CLB: You like Wolfram's writing style?

Knuth: Especially the index... you can easily find your way around that book. With the first edition, when I had a new problem to solve, I would look in the index and it would almost always refer me to the right page. There were three or four times when the word I tried wasn't there, and I penciled in where to look when I had this problem next time. In the

second edition those had all been fixed, and I had not reported them to anybody.

CLB: Let me get your quick impressions on a few research areas, and whether you've read or done any work in them. The first is genetic algorithms. How do you feel about the general concept, that instead of the human determining the algorithm, you somewhat let the machine have at it...

Knuth: I plan to do a lot of experimenting on this as I get into Volume IV. There's genetic breeding, there's simulated annealing, there are other strategies that people have developed. I have a method in The Stanford GraphBase book that I call "stratified greed". These techniques are all competing for the same kind of problems, and I want to try a lot of examples; some of them might work better on one than the other, and I want to get a feel for this. Certain problems are naturals for neural nets... genetic algorithms are likely to do well on tasks related to language recognition, and people say also like predicting the stock market or something like that. Somehow the closer a problem is to nature, the more you expect the genetic algorithm to work, while the closer it is to number theory or something artificial, the more you expect some other kind of approach will help. It's hard to understand the way these methods scale up; on a small problem they might do terrifically, and then they might break down completely just when the problem gets a little bit bigger... or it might go the other way.

CLB: It sounds like you have several years of disciplined testing with your data sets ahead of you.

Knuth: The Stanford GraphBase gives me an unlimited supply of problems that I and other people can do. I read what other people have claimed about their methods, but I also try them all myself. The original work I do in The Art of Computer Programming is to take the methods of two different authors and analyze method A from the standpoint of author B, and method B from the standpoint of author A. They have only given their sides of it, so I try to fill in ....

CLB: What about object-oriented programming? Is it just a current buzzword, or does this approach appeal to you?

Knuth: I've always thought of programming in that way, but I haven't used languages that help enforce the discipline; I've always enforced the discipline myself in other languages. Programming languages can now catch you if you make a mistake, and they make it easier for you to hide information from one part of the program to another. In my own programs, with older languages, I wouldn't use what I wasn't supposed to use; I would have to discipline myself to follow these rules. I could, so I did. There weren't programs I couldn't write... but the new tools do help.

The problem that I have with them today is that... C++ is too complicated. At the moment, it's impossible for me to write portable code that I believe would work on lots of different systems, unless I avoid all exotic features. Whenever the C++ language designers had two competing ideas as to how they should solve some problem, they said "OK, we'll do them both". So the language is too baroque for my taste. But each user of C++ has a favorite subset, and that's fine. CWEB fully supports C++ as well as C.

CLB: What are your thoughts about chaos theory, fractals, those areas? Their indeterminateness seems a little discordant with the domains you've focused on in the past.

Knuth: I did some early work with fractals and so on, and I think it's a great new abstraction. People can build models that they wouldn't have thought of building before, that really match a lot of things in nature that have this character of looking the same when you change the scale. You know, if you magnify the coastline, it still looks like a coastline, and a lot of other things have this property. Nature has recursive algorithms that it uses to generate clouds and Swiss cheese and things like that. So now we have mathematical techniques for understanding such processes that go beyond the kind of differential equations that people used to have in previous centuries. Now we have a brand new tool to work with, but I'm not very intuitive about such methods. I know the limitations of my own intuition; I can solve some problems well, but I know other people are able to see something right away which takes me a long time... It's not my cup of tea.

CLB: To what extent have you ever followed developments in artificial intelligence? The third program you ever wrote was a tic-tac-toe program that learned from its errors, and Stanford has been one of the leading institutions for AI research...

Knuth: Well, AI interacts a lot with Volume IV; AI researchers use the combinatorial techniques that I'm studying, so there is a lot of literature there that is quite relevant. My job is to compare the AI literature with what came out of the electrical engineering community, and other disciplines; each community has had a slightly different way of approaching the problems. I'm trying to read these things and take out the jargon and unify the ideas. The hardest applications and most challenging problems, throughout many years of computer history, have been in artificial intelligence--- AI has been the most fruitful source of techniques in computer science. It led to many important advances, like data structures and list processing... artificial intelligence has been a great stimulation. Many of the best paradigms for debugging and for getting software going, all of the symbolic algebra systems that were built, early studies of computer graphics and

computer vision, etc., all had very strong roots in artificial intelligence.

CLB: So you're not one of those who deprecates what was done in that area...

Knuth: No, no. What happened is that a lot of people believed that AI was going to be the panacea. It's like some company makes only a 15% profit, when the analysts were predicting 18%, and the stock drops. It was just the clash of expectations, to have inflated ideas that one paradigm would solve everything. It's probably true with all of the things that are flashy now; people will realize that they aren't the total answer. A lot of problems are so hard that we're never going to find a real great solution to them. People are disappointed when they don't find the Fountain of Youth...

CLB: If you were a soon-to-graduate college senior or Ph.D. and you didn't have any "baggage", what kind of research would you want to do? Or would you even choose research again?

Knuth: I think the most exciting computer research now is partly in robotics, and partly in applications to biochemistry. Robotics, for example, that's terrific. Making devices that actually move around and communicate with each other. Stanford has a big robotics lab now, and our plan is for a new building that will have a hundred robots walking the corridors, to stimulate the students. It'll be two or three years until we move in to the building. Just seeing robots there, you'll think of neat projects. These projects also suggest a lot of good mathematical and theoretical questions. And high level graphical tools, there's a tremendous amount of great stuff in that area too. Yeah, I'd love to do that... only one life, you know, but...

CLB: Why do you mention biochemistry?

Knuth: There's millions and millions of unsolved problems. Biology is so digital, and incredibly complicated, but incredibly useful. The trouble with biology is that, if you have to work as a biologist, it's boring. Your experiments take you three years and then, one night, the electricity goes off and all the things die! You start over. In computers we can create our own worlds. Biologists deserve a lot of credit for being able to slug it through.

It is hard for me to say confidently that, after fifty more years of explosive growth of computer science, there will still be a lot of fascinating unsolved problems at peoples' fingertips, that it won't be pretty much working on refinements of well-explored things. Maybe all of the simple stuff and the really great stuff has been discovered. It may not be true, but I can't predict an unending growth. I can't be as confident about computer science as I can about biology. Biology easily has 500 years of exciting problems to work on, it's at that level.

CLB: Use of the Internet is exploding right now, with everyone getting on...

Knuth: Some day we are going to try to figure out who is paying for it!

CLB: Do you currently use it? I know you did in the past.

Knuth: I spent fifteen years using electronic mail on the ARPANET and the Internet. Then, in January 1990, I stopped, because it was taking up too much of my time to sift through garbage. I don't have an email address. People trying to write me unsolicited email messages get a polite note saying "Professor Knuth has discontinued reading electronic mail; you can write to him at such and such an address."

It's impossible to shut email off! You send a message to somebody, and they send it back saying "Thank you", and you say "OK, thanks for thanking me..."

Email is wonderful for some people, absolutely necessary for their job, and they can do their work better. I like to say that for people whose role is to be on top of things, electronic mail is great. But my role is to be on the bottom of things. I look at ideas and think about them carefully and try to write them up... I move slowly through things that people have done and try to organize the material. But I don't know what is happening this month.

So now I don't read electronic mail, but I do use it occasionally. Say I'm taking a trip to Israel and I've got to make last minute arrangements. When I visit another university or research center for a few days, I have to send email from there. I've learned how to use the email facilities in Emacs, but I don't want to get good at it.

CLB: You have many interests outside of computing and mathematics--- music, religion, writing. Is music a creative outlet for you, a means of recreation, or a spiritual outlet?

Knuth: At the moment it's recreational. I like to have friends come to the house and play four-hands piano music. If I could do it every week, I would. I hope to live long enough so that after I've finished my life's work on The Art of Computer Programming, I might compose some music. Just a dream... it might be lousy music, of course.

CLB: You have written some compositions already, haven't you?

Knuth: Yeah, but it was mostly arrangements of other people's themes. I did write a short musical comedy when I was in college called "Nebbishland". Remember how Nebbishes were all the rage in the late 50s? "Nebbishland" was only about a ten minute skit, but it was all original music and lyrics.

CLB: Do you have the score somewhere in the attic?

Knuth: Yeah... no actually, I think I've lost it. I have only part of it. I'm hoping to come across it. I'm going through my files now and making a computer index of everything I have in the house.

CLB: Sounds like you don't have a paperless house!

Knuth: No!

CLB: Have you fiddled with MIDI computer technology for music, or have you purposely stayed away from it?

Knuth: I have fun with it. I bought a synthesizer for my son last Christmas, and I played it for hours and hours. I loved it. I had once played on a Kurzweil synthesizer years ago, at Marvin Minsky's house, a grand piano imitation. More recently, a friend went to England for three years and didn't want to bring his grand piano him, so he bought a Yamaha with six voices. When I visited his house, I had a tremendous time for three days going through all of the pieces I'd learned on the piano, playing them as if they were on vibraphone, or on a harpsichord, or some other voice. His "piano" has a harpsichord voice, but the keyboard is pressure-sensitive, so you can play loud and soft, which you can't do on a real harpsichord. These synthesizers are great.

CLB: When did you retire from Stanford?

Knuth: This year. I was on leave for two years until I could officially retire. Unofficially, I retired in 1990, on the same day I gave up email. I announced my plans three years earlier. I realized that my main goal in life was to finish The Art of Computer Programming; I had looked ahead and seen that it would take twenty years of work, full-time. If I continued doing everything else that I was doing, it was going to be forty or fifty years of work. I was just not getting anywhere, I was getting further and further behind. So I said, "Enough." Naturally, I hate to give up many of these other things that I like doing very much. But there are some things I didn't hate giving up, like writing proposals. I'm very happy to give up those!

CLB: You had to write proposals?? I assumed you were insulated from that somehow.

Knuth: You've got a great sense of humor! I don't have to do it anymore; but as a professor, in order to have decent equipment for my grad students, or to have visitors for active research programs, to publish reports, etc., I needed to find sponsors. It's a lot of work begging for money. The System Development Foundation said they'd give me a million dollars so that I could finish TeX and get back to The Art of Computer Programming.

CLB: Did you take them up on it?

Knuth: Sure, but it still took many, many years to finish TeX. I decided that the only way I would be able to finish The Art of Computer Programming is by going into full-time writing, and being a hermit, and telling people "No." It was hard to adjust the first couple of years. Now I feel real efficient, and the writing is going well. A nice steady state.

I give lectures at Stanford every month or so, when I'm in town, called "Computer Musings" . I plan to keep this up for twenty years, to give a talk on whatever I find interesting that month, on neat ideas I've picked up... I bring up problems that I can't solve, so that somebody will do it for me. Now, if I can't solve a problem in two hours, I've got to give it up and tell somebody else to work on it; otherwise, I'll get behind again. As I write the book, I've got to move from topic to topic, and my attention span is maybe three weeks on any particular topic.

CLB: You're best known for your writing and research; did you enjoy teaching and the interaction with students?

Knuth: We had the greatest students in the world. I can still get together with students through my lecture series, except I don't know their names anymore. That's a problem.

CLB: No student interns?

Knuth: Suppose I give a "Computer Musings" lecture, stating an open problem, and suppose that a student in the

audience solves that problem, writes his thesis and finishes it in the next two weeks (maybe two and a half), and shows it to me. Then I'd still be interested in the topic, would still read it, and I'd be glad to sign his thesis... but that's the only way. 28 is the total number of Ph.D. students I've had graduate, and that's probably all that I will have... unless something happens at high speed through the "Computer Musings".

CLB: Real-time Ph.D.'s! What changes have you seen in the students coming into the computer science program over the years?

Knuth: There is a very profound change that I can't account for. In the 70s, the majority of our students were very interested in music. The first thing we'd ask them when they came in was "What instrument do you play?" We had lots of chamber groups and so on. Now almost none of the students are interested in music. I don't know if it's because a different kind of people are enrolling in computer science, or because it's true of all today's students, or what. If you ask computer science students now what their hobby is, the chances are most of them will say "Bicycling". I recently had one who played a harmonica, but there were almost no musicians in the group.

CLB: Any changes in the quality of the students?

Knuth: Not the quality... but they don't know as much about mathematics as they used to. We have to do more remedial stuff in college, even at a school like Stanford.

CLB: How about changes in the field itself... with so much progress and so many more people involved, is computer science today very different than it was earlier?

Knuth: Well, there's all the media and the visual things, that's a lot different than it was. There's also the competition; it's a great deal more difficult now than it was in my day. When I started, it was so easy to come up with something new compared to now, when you've got thousands and thousands of smart people all doing great stuff. There might have been ten great Ph.D. theses a year at one time; there's just no way to keep up with all the stuff now.

No matter what field of computer science you're in, everyone is finding it hard to keep up. Every field gets narrower and narrower, since nobody can cover all the territory anymore. Everybody can choose two small parts of computer science and learn those two parts; if one person knows parts A and B, another knows B and C, and another knows C and D, the field remains reasonably well connected, even as it expands.

CLB: Do you see yourself as one of the last of computer science's "Renaissance Men"?

Knuth: I'm not as broad as you might think--- I only work on one thing at a time. I guess I'm a quick study; I can become an instant expert in something. I've been collecting stuff for thirty years so that I can read the literature on each topic in "batch mode"--- not swapping lots of different topics in and out. I can absorb a subject locally and get good at it for a little while... but then don't ask me to do the thing I was doing a few months ago! Also, I have lots of people helping me correct my mistakes.

CLB: My last question, your least favorite to be asked... what is the current plan for completing all seven volumes of The Art of Computer Programming?

Knuth: I'm going to have fascicles of about 128 pages coming out twice a year. We're gathering four of them before we come out with the first two actually; we're going to keep some in the pipeline! Look for the first fascicles in 1995 or 1996; they will be beta-test versions of the real books. I'm thinking I can finish Volume IV (parts A, B, and C)in the year 2003, Volume V in 2008, then come out with new editions of Volume I, II, and III, then work on VI and VII... There will be a "Reader's Digest" version of volumes I through V.

CLB: What would your career, and life, have been like had you not announced the 7-volume set?

Knuth: Oh, I didn't announce it at first. I thought I was writing only one book. But if I hadn't done that, I suppose I still would have been doing a lot of writing. Somehow it seems that all the way through, I've enjoyed trying to explain things. When I was in high school, I was editor of the student paper; in college I edited a magazine. I've always been playing around with words.