viewpoints

DOI: 10.1145/1364782.1364794

Len Shustek

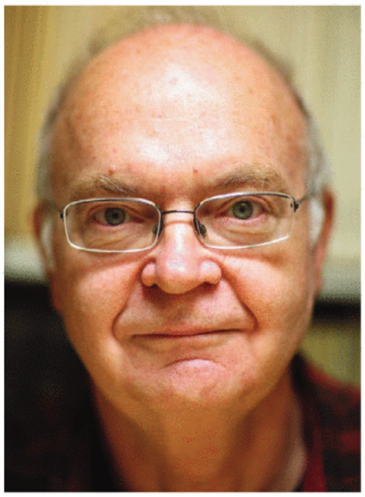
Interview
The 'Art' of Being Donald Knuth

In this first of a two-part talk, the renowned scholar and computer scientist reflects on the influences that set the course for his extraordinary career.

THE COMPUTER HISTORY Museum has an active program to gather videotaped histories from people who have done pioneering work in this first century of the information age. These tapes are a rich aggregation of stories that are preserved in the collection, transcribed, and made available on the Web to researchers, students, and anyone curious about how invention happens.

The oral histories are conversations about people's lives. We want to know about their upbringing, their families, their education, and their jobs. But above all, we want to know how they came to the passion and creativity that leads to innovation.

Presented here in two installments (concluding next month) are excerpts[a] from an interview conducted by Edward Feigenbaum in March 2007 of Donald E. Knuth, Professor Emeritus of The Art of Computer Programming at Stanford University. —L. S.



PHOTOGRAPH BY TIMOTHY ARCHIBALD

Don talks about his family background.

My father was the first person among all his ancestors who had gone to college. My mother was the first person in all of her ancestors who had gone to a year of school to learn how to be a typist. My great-grandfather was a blacksmith. There was no tradition in our family of higher education at all. These people were pretty smart, but they didn't have an academic background.

Some people know from an early age what they want to do. Don didn't, but he knew he wanted to work hard.

My main interest in those days was music. But at the college where I had been admitted, people emphasized how easy it was going to be there as a music major. When I got the chance to go to Case Institute of Technology in Ohio instead, I was intrigued by the idea that Case was going to make me work hard. I was scared that I was going to flunk out, but still I was ready to work.

He initially aspired to be a physicist, but something happened along the way.

In my sophomore year in physics I had to take a required class of welding. Welding was so scary and I was a miserable failure at it, so I decided maybe I can't be a physicist. On the other hand—mathematics! In the sophomore year for mathematicians, they give you courses on what we now call discrete mathematics, where you study logic and things that are integers instead of continuous quantities. I was drawn to that. That was something, somehow, that had great appeal to me.

I think that there is something strange inside my head. It's clear that I have much better intuition about discrete things than continuous things. In physics, for example, I could pass the exams and I could do the problems in quantum mechanics, but I couldn't intuit what I was doing. But on the other hand, in my discrete math class, these were things that really seemed a part of me. There's definitely something in how I had developed by the time I was a teenager that made me understand discrete objects, like zeros and ones of course, or things that are made out of alphabetical letters, much better than things like Fourier transforms or waves.

I'm visualizing the symbols. To me, the symbols are reality, in a way. I take

a Oral histories are not scripted, and a transcript of casual speech is very different from what one would write. I have taken the liberty of editing and reordering freely for presentation. For the original transcript, see http://archive.computerhistory.org/search/oh/

My first program taught me a lot about the errors that I was going to be making in the future, and also about how to find errors. That's sort of the story of my life, making errors and trying to recover from them. I try to get things correct. I probably obsess about not making too many mistakes.

a mathematical problem, I translate it into formulas, and then the formulas are the reality.

He discovers computers, and how hard programming is.

I wrote my first program for the IBM 650 [a vacuum tube magnetic drum computer from the 1950s], probably in the spring of my freshman year, and debugged it at night. The first time I wrote the program, to find the prime factors of a number, it was about 60 instructions long in machine language. They were almost all wrong. When I finished, it was about 120 or 130 instructions. I made more errors in this program than there were lines of code!

My first program taught me a lot about the errors that I was going to be making in the future, and also about how to find errors. That's sort of the story of my life, making errors and trying to recover from them. I try to get things correct. I probably obsess about not making too many mistakes.

At Case he learns about early compilers

For the IT ("Internal Translator") program for the 650 you would punch an algebraic formula on cards and feed the cards into the machine. The lights spin around for a few seconds and then out come machine language instructions that set X1 equal to X2 + X4. Automatic programming coming out of an algebraic formula! Well, this blew my mind. I couldn't understand how it was possible to do this miracle. I could understand how to write a program to factor numbers, but I couldn't understand how to write a program that would convert algebra into machine instructions.

It hadn't yet occurred to him that the computer was a general symbol-manipulating device?

No. That occurred to Lady [Ada] Lovelace, but it didn't occur to me. I'm slow to pick up on these things, but then I persevere.

I got hold of the source code for IT. I went through every line of that program. During the summer we typically had a family get-together on a beach on Lake Erie where we spent time playing cards and playing tennis. But that summer, I spent most of the time going through this listing, trying to find out the miracle of how IT worked. Okay, it wasn't impossible after all. In fact, I thought of better ways to do it than were in that program.

The code, once I saw how it happened, was inspiring to me. Also, the discipline of reading other people's programs was something good to learn early. Throughout my life I've had a love of reading source materials—reading something that pioneers had written and trying to understand their thought processes, especially when they're solving a problem I don't know how to solve. This is the best way for me to get my own brain past the stumbling blocks. At Case I remember looking at papers that [Pierre de] Fermat had written in Latin in the 17th century, in order to understand how that great number theorist approached problems.

But it's been hard to communicate the love of reading historical programs.

I would say that's my major disappointment with my teaching career. I was not able to get across to any of my students this love for that kind of

scholarship—reading source material. I was a complete failure at passing this on to the people that I worked with the most closely.

He graduates from Case and becomes a professional compiler writer while traveling to the California Institute of Technology for graduate school.

I had learned about the Burroughs 205 machine language, and it was kind of appealing to me. So I made my own

PHOTOGRAPH BY TIMOTHY ARCHIBALD

proposal to Burroughs. I said, "I'll write you an ALGOL compiler for $5,000. But I can't implement all of ALGOL for this; I am just one guy. Let's leave out procedures." Well, this is a big hole in the language! Burroughs said, "No, you've got to put in procedures." I said, "Okay, I will put in procedures, but you've got to pay me $5,500." That's what happened. They paid me $5,500, which was a fairly good salary in those days. So between graduating from Case and going to Caltech, I worked on this compiler. Heading out to California, I drove 100 miles each day and then sat in a motel and wrote code.

But he rejects "compiler writer" as a career, and decides what is important in life.

Then a startup company came to me and said, "Don, write compilers for us and we will take care of finding computers to debug them. Name your price." I said, "Oh, okay, $100,000," assuming that this was [outrageous]. The guy didn't blink. He agreed. I didn't blink either. I said, "I'm not going to do it. I just thought that was an impossible number." At that point I made the decision in my life that I wasn't going to optimize my income.

I spent a day that summer looking at the mathematics of how fast linear probing works. I got lucky, and I solved the problem. I figured out some math, and I kept two or three sheets of paper with me and I typed it up.[b] This became the genesis of my main research work, which developed not to be working on compilers, but to be working on the analysis of algorithms. It dawned on me that this was just one of many algorithms that would be important, and each one would lead to a fascinating mathematical problem. This was easily a good lifetime source of rich problems to work on.

If you ask me what makes me most happy, number one would be somebody saying "I learned something from you." Number two would be somebody saying "I used your software."

If you ask me what makes me most happy, number one would be somebody saying "I learned something from you." Number two would be somebody saying "I used your software."

At Caltech he finds a mentor, but can't talk to him.

I went to Caltech because they had [strength] in combinatorics, although their computing system was incredibly arcane and terrible. Marshall Hall was my thesis advisor. He was a world-class mathematician, and for a long time had done pioneering work in combinatorics. He was my mentor. But it was a funny thing, because I was in such awe of him that when I was in the same room with him I could not think straight. I wouldn't remember my name. I would write down what he was saying, and then I would go back to my office so that I could figure it out. We couldn't do joint research together in the same room. We could do it back and forth.

He also was an extremely good advisor, in better ways than I later was with my students. He would keep track of me to make sure I was not slipping. When I was working with my own graduate students, I was pretty much in a mode where they would bug me instead of me bugging them. But he would actually write me notes and say, "Don, why don't you do such and such?"

The research for his Ph.D. thesis takes an hour.

I got a listing from a guy at Princeton who had just computed 32 solutions to a problem that I had been looking at for a homework problem in my combinatorics class. I was riding up on the elevator with Olga Todd, one of our professors, and I said, "Mrs. Todd, I think I'm going to have a theorem in an hour. I am going to psyche out the rule that explains why there happen to be 32 of each kind." Sure enough, an hour later I had seen how to get from each solution on the first page to the solution on the second page. I showed this to Marshall Hall. He said, "Don, that's your thesis. Don't worry about this block design with $\lambda=2$ business. Write this up instead and get out of here." So that became my thesis. And it is a good thing, because since then only one more design with $\lambda=2$ has been discovered in the history of the world. I might still be working on my thesis if I had stuck to that problem. But I felt a little guilty that I had solved my Ph.D. problem in one hour, so I dressed it up with a few other chapters of stuff.

He's never had trouble finding problems to work on.

The way I work it's a blessing and a curse that I don't have difficulty thinking of questions. I have to actively suppress stimulation so that I'm not working on too many things at once. The hard thing for me is not to find a problem, but to find a good problem. One that will not just be isolated to something that happens to be true, but also will be something that will have spin-offs, so that once you've solved the problem, the techniques are going to apply to many other things.

He starts *The Art of Computer Programming*.

A man from Addison-Wesley came to visit me and said "Don, we would like you to write a book about how to write compilers." I thought about it and decided "Yes, I've got this book inside of me." That day I sketched out—I still have that sheet of tablet paper—12 chapters that I thought should be in such a book. I told my new wife, Jill, "I think I'm going to write a book." Well, we had just four months of bliss, because the rest of our marriage has all been devoted to this book. We still have had happiness, but really, I wake up every morning and I still haven't finished the book. So I try to organize the rest of my life around this, as one main unifying theme.

George Forsythe [founder of the Computer Science Department at Stanford] came down to southern California for a talk, and he said, "Come up to Stanford. How about joining our faculty?" I said "Oh no, I can't do that. I just got married, and I've got to finish this book first. I think I'll finish the book next year, and then I can come up [and] start thinking about the rest of my life. But I want to get my book done before my son is born." Well, John is now 40-some years old and I'm not done with the book.

This is really the story of my life, because I hope to live long enough to finish it. But I may not because it's turned out to be such a huge project.

1967 was a big year.

It was certainly a pivotal year in my life. You can see in retrospect why I think things were building up to a crisis, because I was just working at high pitch all the time. I was on the editorial board of *Communications of the ACM* and *Journal of the ACM*—working on their programming languages sections—and I took the editorial duties very seriously. I was a consultant to Burroughs on innovative machines. I was consumed with getting *The Art of Computer Programming* done. And I was a father and husband. I would start out every day saying "Well, what am I going to accomplish today?" Then I would stay up until I finished it.

It was time for me to make a career decision. The question was where should I spend the rest of my life?

b "Notes on Open Addressing." Unpublished memorandum, July 22, 1963; but see http://algo.inria.fr/AofA/Research/11-97.html

Should I be a mathematician? Should I be a computer scientist? By this time I had learned that it was actually possible to do mathematical work as a computer scientist. I had analysis of algorithms to do. What would be a permanent home? My model of my life was going to be that I was going to make one move in my lifetime to a place where I had tenure, and I would stay there forever.

The crisis comes.
At Caltech, I was preparing my class lectures, or typing my book. I didn't have time to do research. If I had a new idea, if I said "Here's a problem that ought to be solved," when was I going to solve it? Maybe on the airplane. We were doing a lot of experiments but I didn't have time to sit down at home and work out the theory for it. I had attribute grammars coming up in February, and these reductions systems coming up in March, and I was supposed to be grinding out Volume Two of *The Art of Computer Programming*. I was scheduled in June to lecture at a summer school in Copenhagen about how to parse, what's called top-down parsing.

What happened then, in May, is I had a massive bleeding ulcer, and I was hospitalized. My body gave out. I was just doing all this stuff, and it couldn't take it.

I learned about myself. The doctor showed me his textbook that described the typical ulcer patient: what people call the "Type A" personality. It described me to a T. All of the signs were there. I was an automaton, I think, basically. I saw a goal and I put myself to it, and I worked on it and pushed it through. I didn't say no to people when they asked, "Don, can you do this for me?" At this point I saw I had this problem. I shouldn't try to do the impossible.

He changes his lifestyle, and moves to Stanford.
I wrote a letter to my publisher, framed in black, saying, "I'm not going to be able to get the manuscript of Volume Two to you this year. I'm sorry." I resigned from 10 editorial boards. No more JACM, no more CACM. I gave up all of the editorships in order to cut down my workload. I started working on Volume Two where I left off at the time of the ulcer, but I would be careful to go to sleep and keep a regular schedule. I went to a conference in Santa Barbara on combinatorial mathematics and had three days to sit on the beach and develop the theory of attribute grammars, this idea of top-down and bottom-up parsing.

In February of 1968 I finally got the offer from Stanford. The committees were saying, "This guy is just 30 years old." But when they looked at the book, they said, "Oh, there's some credibility here." That helped me.

Why he writes his books with a pencil.
I love keyboards, but my manuscripts are always handwritten. The reason is that I type faster than I think. There's a synchronization problem. I can think of ideas at about the rate I can write them down with a pencil. But with typing I'm going faster, so I have to sync, and my thoughts have to start up and stop again in a way that involves more of my brain.

Three volumes of "The Art" are done, but it's time for a pause.
Volume Four is about combinatorial algorithms. Combinatorial algorithms were such a small topic in 1962, when I made that Chapter Seven of my outline, that Johan Dahl asked me, "How did you ever think of putting in a chapter about combinatorial algorithms in 1962?" I said, "Well, the only reason was that it was the part I thought was most fun." But there was almost nothing known about it at the time.

The way I look at it, this is where you've got to use some art. You've got to be really skillful, because one good idea can save you six orders of magnitude and make your program run a million times faster. People are coming up with these ideas all the time. For me, the combinatorial explosion was the explosion of research in combinatorics. Not the problems exploding, but the ideas were exploding. There's that much more to cover now.

It's true that in the back of my mind I was scared stiff that I can't write Volume Four anymore. So maybe I was waiting for it to simmer down. Somebody did say to me once, after I solved the problem of typesetting, maybe I would start to look at binding or something, because I had to have some other reason [to delay]. I've certainly seen enough graduate student procrastinators in my life. Maybe I was in denial. ■

I told my new wife, Jill, "I think I'm going to write a book." Well, we had just four months of bliss, because the rest of our marriage has all been devoted to this book. We still have had happiness, but really, I wake up every morning and I still haven't finished the book. So I try to organize the rest of my life around this, as one main unifying theme.

He solves the problem of typesetting? Stay tuned for Part II of this interview in the August issue and learn how Knuth interrupted his life's work on The Art of Computer Programming *to create a system that makes digitally produced books beautiful.*

Edited by **Len Shustek**, Chair, Computer History Museum

viewpoints

DOI:10.1145/1378704.1378715 Len Shustek, Editor

Interview Donald Knuth: A Life's Work Interrupted

In this second of a two-part interview by Edward Feigenbaum, we find Knuth, having completed three volumes of The Art of Computer Programming, *drawn to creating a system to produce books digitally.*

Don switches gears and for a while becomes what Ed Feigenbaum calls "The World's Greatest Programmer."

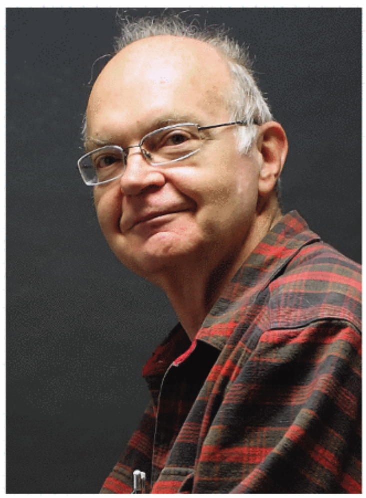
There was a revolutionary new way to write programs that came along in the 1970s called "structured programming." At Stanford we were teaching students how to write programs, but we had never really written more than textbook code ourselves in this style. Here we are, full professors, telling people how to do it, but having never done it ourselves except in really sterile cases with no real-world constraints. I was itching to do it. Thank you for calling me the world's greatest programmer—I was always calling myself that in my head. I love programming, and so I loved to think that I was doing it as well as anybody. But the fact is the new way of programming was something that I hadn't had time to invest much effort in.

The motivation is his love affair with books...

That goes very deep. My parents disobeyed the conventional wisdom by teaching me to read before I entered kindergarten. I have a kind of strange love affair with books going way back. I also had this thing about the appearance of books. I wanted my books to have an appearance that other readers would treasure, not just appreciate because there were some words in there.

For Part I of this interview, see *Communications*, July 2008, page 35.

PHOTOGRAPH BY TIMOTHY ARCHIBALD



...and what had happened to his books.

Printing was done with hot lead in the 1960s, but they switched over to using film in the 1970s. My whole book had been completely re-typeset with a different technology. The new fonts looked terrible! The subscripts were in a different style from the large letters, for example, and the spacing was very bad. You can look at books printed in the early 1970s and almost everything looked atrocious in those days. I couldn't stand to see my books so ugly. I spent all this time working on them, and you can't be proud of something that looks hopeless. I was tearing out my hair.

At the very same time, in February 1977, Pat Winston had just come out with a new book on artificial intelligence, and the proofs of it were being done at III [Information International, Incorporated] in Southern California. They had a new way of typesetting using lasers. All digital, all dots of ink. Instead of photographic images and lenses, they were using algorithms, bits. I looked at Winston's galley proofs. I knew it was just bits, but they looked gorgeous.

I canceled my plan for a sabbatical in Chile. I wrote saying "I'm sorry; instead of working on Volume 4 during my sabbatical, I'm going to work on typography. I've got to solve this problem of getting typesetting right. It's only zeros and ones. I can get those dots on the page, and I've got to write this program." That's when I became an engineer. I did sincerely believe that it was only going to take me a year to do it.

But, in fact, it was to be a 10-year project. The prototype user was Phyllis Winkler, Don's secretary.

Phyllis had been typing all of my technical papers. I have never seen her equal anywhere, and I've met a lot of really good technical typists. My thought was definitely that this would be something that I would make so that Phyllis would be able to take my handwritten manuscripts and go from there.

The design took place in two all-nighters. I made a draft. I sat up at the AI lab one evening and into the early morning hours, composing what I thought would be the specifications of a language. I looked at my book and I found excerpts from several dozen pages where I thought it gave all the variety of things I need in the book. Then I sat down and I thought, well, if I were Phyllis, how would I like to key this in? What would be a reasonable format that would appeal to Phyllis, and at the same time something that as a compiler writer I felt I could translate into the book? Because TeX is just another kind of a compiler; instead of going into machine language you're going into words on a page. That's a different output language, but it's analogous to recognizing the constructs that appear in the source file.

The programming turned out to be harder than he thought.

I showed the second version of the design to two of my graduate students, and I said, "Okay, implement this, please, this summer. That's your summer job." I thought I had specified a language. To my amazement, the students, who were outstanding students, did not complete it. They had a system that was able to do only about three lines of TeX. I thought, "My goodness, what's going on? I thought these were good students." Later I changed my attitude, saying, "Boy, they accomplished a miracle." Because going from my specification, which I thought was complete, they really had an impossible task, and they had succeeded wonderfully with it. These guys were actually doing great work, but I was amazed that they couldn't do what I thought was just sort of a routine task. Then I became a programmer in earnest, I had to do it.

This experience led to general observations about programming and specifications.

When you're doing programming, you have to explain something to a computer, which is dumb. When you're writing a document for a human being to understand, the human being will look at it and nod his head and say, "Yeah, this makes sense." But there are all kinds of ambiguities and vagueness that you don't realize until you try to put it into a computer. Then all of a sudden, almost every five minutes as you're writing the code, a question comes up that wasn't addressed in the specification. "What if this combination occurs?" It just didn't occur to the person writing the design specification. When you're faced with doing the implementation, a person who has been delegated the job of working from a design would have to say, "Well, hmm, I don't know what the designer meant by this."

It's so hard to do the design unless you're faced with the low-level aspects of it, explaining it to a machine instead of to another person. I think it was George Forsythe who said, "People have said you don't understand something until you've taught it in a class. The truth is you don't really understand something until you've taught it to a computer, until you've been able to program it." At this level, programming was absolutely important.

When I got to actually programming TeX, I had to also organize it so that it could handle lots of text. I had to develop a new data structure in order to be able to do the paragraph coming in text and enter it in an efficient way. I had to introduce ideas called "glue," and "penalties," and figure out how that glue should disappear at boundaries in certain cases and not in others. All these things would never have occurred to me unless I was writing the program.

> **"I wake up in the morning with an idea, and it makes my day to think of adding a couple of lines to my program. It gives me a real high. It must be the way poets feel, or musicians, or painters. Programming does that for me."**

Edsger Dijkstra gave this wonderful Turing lecture early in the 1970s called "The Humble Programmer." One of the points he made in his talk was that when they asked him in Holland what his job title was, he said, "Programmer," and they said, "No, that's not a job title. You can't do that; programmers are just coders. They're people who are assigned like scribes were in the days when you needed somebody to write a document in the Middle Ages." Dijkstra said no, he was proud to be a programmer. Unfortunately, he changed his attitude completely, and I think he wrote his last computer program in the 1980s.

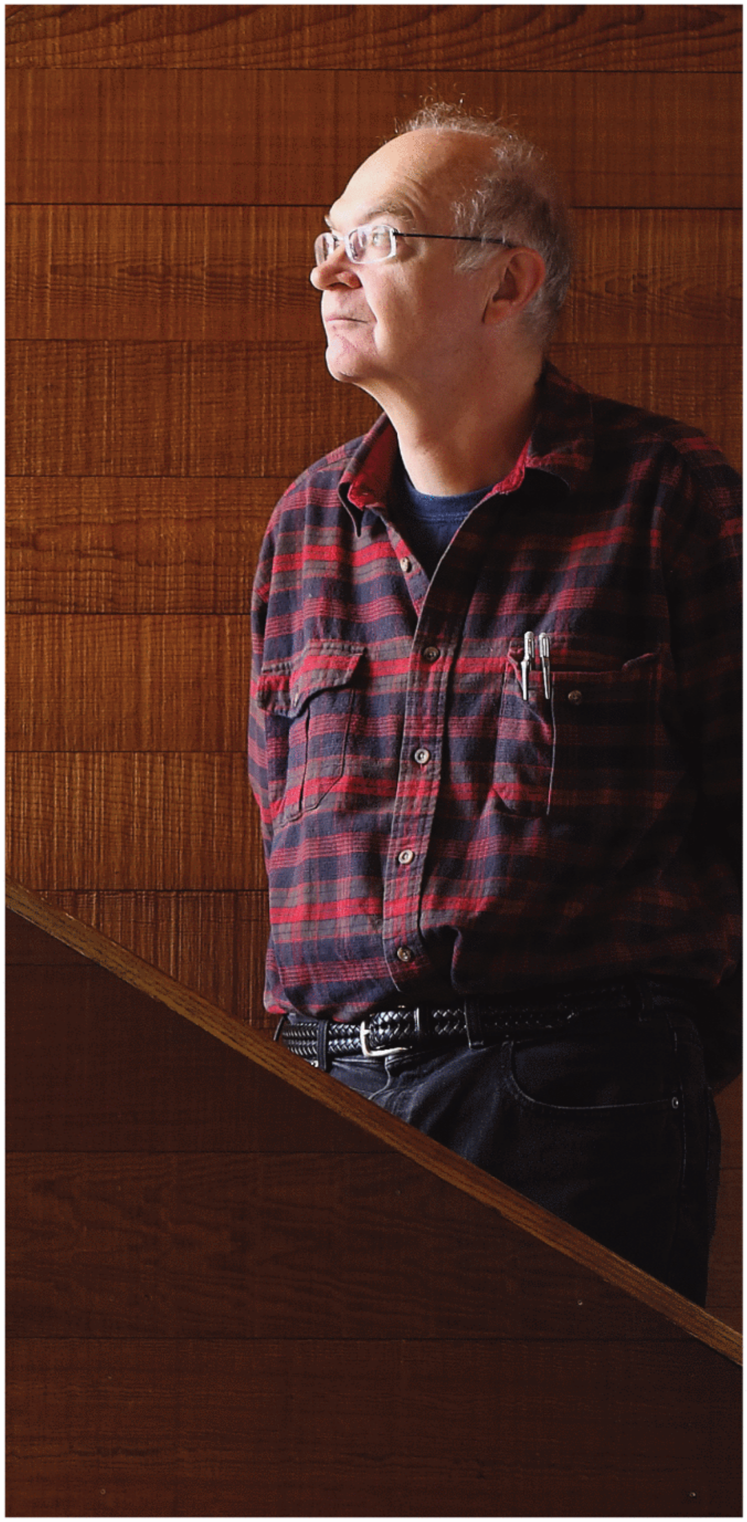
I checked the other day and found I wrote 35 programs in January, and 28 or 29 programs in February. These are small programs, but I have a compulsion. I love to write programs. I think of a question that I want to answer, or I have part of my book where I want to present something, but I can't just present it by reading about it in a book. As I code it, it all becomes clear in my head. The fact that I have to translate my knowledge of this method into something that the machine is going to understand forces me to make that knowledge crystal-clear in my head. Then I can explain it to somebody else infinitely better. The exposition is always better if I've implemented it, even though it's going to take me more time.

It didn't occur to me at the time that I just had to program in order to be a happy man. I didn't find my other roles distasteful, except for fundraising. I enjoyed every aspect of being a professor except dealing with proposals, which was a necessary evil. But I wake up in the morning with an idea, and it makes my day to think of adding a couple of lines to my program. It gives me a real high. It must be the way poets feel, or musicians, or painters. Programming does that for me.

The TeX project led to METAFONT for the design of fonts. But it also wasn't smooth sailing.

Graphic designers are about the nicest people I've ever met in my life. In

"I found that writing software was much more difficult than anything else I had done in my life. I had to keep so many things in my head at once. I couldn't just put it down and start something else. It really took over my life during this period."



PHOTOGRAPH BY TIMOTHY ARCHIBALD

the spring of 1977, I could be found mostly in the Stanford Library reading about the history of letter forms. Before I went to China that summer I had drafted the letters for A to Z.

One of the greatest disappointments in my whole life was the day I received in the mail the new edition of *The Art of Computer Programming* Volume 2, which was typeset with my fonts and which was supposed to be the crowning moment of my life, having succeeded with the TeX project. I think it was 1981, and I had the best typesetting equipment, and I had written a program for the 8-bit microprocessor inside. It had 5,000 dots-per-inch, and all the proofs coming out looked good on this machine. I went over to Addison-Wesley, who had typeset it. There was the book, and it was in the familiar beige covers. I opened the book up and I'm thinking, "Oh, this is going to be a nice moment." I had Volume 2, first edition. I had Volume 2, second edition. They were supposed to look the same. Everything I had known up to that point was that they would look the same. All the measurements seemed to agree. But a lot of distortion goes on, and our optic nerves aren't linear. All kinds of things were happening. I burned with disappointment. I really felt a hot flash, I was so upset. It had to look right, and it didn't, at that time. I'm happy to say that I open my books now and I like what I see. Even though they don't match the 1968 book exactly, the way they differ are pleasing to me.

What it was like writing TeX.

Structured programming gave me a different feeling from programming the old way—a feeling of confidence that I didn't have to debug something immediately as I wrote it. Even more important, I didn't have to mock-up the unwritten parts of the program. I didn't have to do fast prototyping or something like that, because when you use structured programming methodology you have more confidence that it's going to be right, that you don't have to try it out first. In fact, I wrote all of the code for TeX over a period of seven months, before I even typed it into a computer. It wasn't until March 1978 that I spent three weeks debugging everything I had written up to that time.

I found that writing software was much more difficult than anything else I had done in my life. I had to keep so many things in my head at once. I couldn't just put it down and start something else. It really took over my life during this period. I used to think there were different kinds of tasks: writing a paper, writing a book, teaching a class, things like that. I could juggle all of those simultaneously. But software was an order of magnitude harder. I couldn't do that and still teach a good Stanford class. The other parts of my life were largely on hold, including *The Art of Computer Programming*. My life was pretty much typography.

TeX leads to a new way of programming.

Literate programming, in my mind, was the greatest spin-off of the TeX project. I learned a new way to program. I love programming, but I really love literate programming. The idea of literate programming is that I'm writing a program for a human being to read rather than a computer to read. It's still a program and it's still doing the stuff, but I'm a teacher to a person. I'm addressing my program to a thinking being, but I'm also being exact enough so that a computer can understand it as well. Now I can't imagine trying to write a program any other way.

As I'm writing *The Art of Computer Programming*, I realized the key to good exposition is to say everything twice: informally and formally. The reader gets to lodge it in his brain in two different ways, and they reinforce each other. In writing a computer program, it's also natural to say everything in the program twice. You say it in English, what the goals of this part of the program are, but then you say it in your computer language. You alternate between the informal and the formal. Literate programming enforces this idea.

In the comments you also explain what doesn't work, or any subtleties. You can say, "Now note the following. Here is the tricky part in line 5, and it works because of this." You can explain all of the things that a maintainer needs to know. All this goes in as part of the literate program, and makes the program easier to debug, easier to maintain, and better in quality.

After TeX, Don gets to go back to mathematics.

We finished the TeX project; the climax was in 1986. After a sabbatical in Boston I came back to Stanford and plunged into what I consider my main life's work: analysis of algorithms. That's a very mathematical thing, and so instead of having font design visitors to my project, I had great algorithmic analysts visiting my project. I started working on some powerful mathematical approaches to analysis of algorithms that were unheard of in the 1960s when I started the field. Here I am in math mode, and thriving on the beauties of this subject.

"At age 55 I became 'Professor Emeritus of The Art of Computer Programming,' with a capital 'T.' I love that title."

One of the problems out there that was fascinating is the study of random graphs. Graphs are one of the main focuses of Volume 4, all the combinatorial algorithms, because they're ubiquitous in applications.

Frustrated with the rate of progress, he "retires" to devote himself to "The Art."

I wasn't really as happy as I let on. I mean, I was certainly enjoying the research I was doing, but I wasn't making any progress at all on Volume 4. I'm doing this work on random graphs, and I'm learning all of these things. But at the end of the year, how much more had been done? I've still got 11 feet of preprints stacked up in my closet that I haven't touched, because I had to put that all on hold for the TeX project. I figured the thing that I'm going to be able to do best for the world is finishing *The Art of Computer Programming*.

The only way to do it was to stop being a professor full time. I really had to be a writer full time. So, at age 55 I became "Professor Emeritus of The Art of Computer Programming," with a capital "T." I love that title.

Don is a master at straddling the path between engineering and science.

I always thought that the best way to sum up my professional work is that it has been an almost equal mix of theory and practice. The theory I do gives me the vocabulary and the ways to do practical things that can make giant steps instead of small steps when I'm doing a practical problem. The practice I do makes me able to consider better and more robust theories, theories that are richer than if they're just purely inspired by other theories. There's this symbiotic relationship between those things. At least four times in my life when I was asked to give a kind of philosophical talk about the way I look at my professional work, the title was "Theory and Practice." My main message to the theorists is, "Your life is only half there unless you also get nurtured by practical work."

Software is hard. My experience with TeX taught me to have much more admiration for colleagues that are devoting most of their life to software than I had previously done, because I didn't realize how much more bandwidth of my brain was being taken up by that work than it was when I was doing just theoretical work.

Computers aren't everything: religion is part of his life, too.

I think computer science is wonderful, but it's not everything. Throughout my life I've been in a very loving religious community. I appreciate Luther as a theologian who said you don't have to close your mind. You keep questioning. You never know the answer. You don't just blindly believe something.

I'm a scientist, but on Sundays I would study with other people of our church on aspects of the Bible. I got this strange idea that maybe I could study the Bible the way a scientist would do it, by using random sampling. The rule I decided on was we were going to study Chapter 3, Verse 16 of every book of the Bible.

This idea of sampling turned out to be a good time-efficient way to get into a complicated subject. I actually got too confident that I knew much more than I actually had any right to, because I'm only studying less than 1/500th of the Bible. But a classical definition of a liberal education is that you know everything about something and something about everything.[a]

On his working style...

I enjoy working with collaborators, but I don't think they enjoy working with me, because I'm very unreliable. I march to my own drummer, and I can't be counted on to meet deadlines because I always underestimate things. I'm not a great coworker, and I'm very bad at delegating.

I have no good way to work with somebody else on tasks that I can do myself. It's a huge skill that I lack. With the TeX project I think it was important, however, that I didn't delegate the writing of the code. I needed to be the programmer on the first-generation project, and I needed to write the manual, too. If I delegated that, I wouldn't have realized some parts of it are impossible to explain. I just changed them as I wrote the manual.

a See *3:16 Bible Texts Illuminated*, by Donald Knuth, A-R Editions, 1991.

"I'm worried about the present state of programming. Programmers now are ... supposed to assemble reusable code that somebody else has written... Where's the fun in that? Where's the beauty in that?"

What is the future of programming?

A program I read when I was in my first year of programming was the SOAP II assembler by Stan Poley at IBM. It was a symphony. It was smooth. Every line of code did two things. It was like seeing a grand master playing chess. That's the first time I got a turn-on saying, "You can write a beautiful program." It had an important effect on my life.

I'm worried about the present state of programming. Programmers now are supposed to mostly just use libraries. Programmers aren't allowed to do their own thing from scratch anymore. They're supposed to assemble reusable code that somebody else has written. There's a bunch of things on the menu and you choose from these and put them together. Where's the fun in that? Where's the beauty in that? We have to figure out a way we can make programming interesting for the next generation of programmers.

What about the future of science and engineering generally?

Knowledge in the world is exploding. Up until this point we had subjects, and a person would identify themselves with what I call the vertices of a graph. One vertex would be mathematics. Another vertex would be biology. Another vertex would be computer science, a new one. There would be a physics vertex, and so on. People identified themselves as vertices, because these were the specialties. You could live in that vertex, and you would be able to understand most of the lectures that were given by your colleagues.

Knowledge is growing to the point where nobody can say they know all of mathematics, certainly. But there's so much interdisciplinary work now. We see that a mathematician can study the printing industry, and some of the ideas of dynamic programming apply to book publishing. Wow! There are interactions galore wherever you look. My model of the future is that people won't identify themselves with vertices, but rather with edges—with the connections between. Each person is a bridge between two other areas, and they identify themselves by the two subspecialties that they have a talent for.

Finally, we always ask for life advice.

When I was working on typography, it wasn't fashionable for a computer science professor to do typography, but I thought it was important and a beautiful subject. Other people later told me that they're so glad I put a few years into it, because it made it academically respectable, and now they could work on it themselves. They were afraid to do it themselves. When my books came out, they weren't copies of any other books. They always were something that hadn't been fashionable to do, but they corresponded to my own perception of what ought to be done.

Don't just do trendy stuff. If something is really popular, I tend to think: back off. I tell myself and my students to go with your own aesthetics, what you think is important. Don't do what you think other people think you want to do, but what you really want to do yourself. That's been a guiding heuristic for me all the way through.

And it should for the rest of us. Thank you, Don.

Edited by **Len Shustek**, Chair, Computer History Museum, Mountain View, CA.