Crossroads
The ACM Student Magazine

Stephen R. Schach

Saveen Reddy

Stephen R. Schach is an Associate Professor of Computer Science in Vanderbilt University's Computer Science Department. He earned his M.Sc. at the Weizmann Institute of Science in Israel and his Ph.D. at the University of Cape Town, South Africa. He is the author of more than seventy papers on topics such as software engineering, software testing, and computer architecture and has written several books on software engineering. He also consults for the industry and teaches courses on software engineering internationally.

Can you tell me a little bit about your origins?

Yes, I was born in Cape Town, South Africa in 1947, a fourth generation South African. Throughout my childhood, my parents taught us that the racial system there was wrong and unacceptable, but that neither I nor my sister had the right to leave. We had to stay there in order to do something about the situation which had to be opposed, but the way to oppose it was not simply to leave. However, the government made opposition extremely difficult. If you crossed a certain unwritten line, you were simply thrown into jail as a communist - by definition someone opposed to government

policies. By 1982, I came to the conclusion that there was nothing more that people of goodwill could do. By that stage, my children were five and two years old, and I didn't want to bring them up in a racist society. I had managed to fight the virus of racism, but maybe my children might become infected. So, my wife and I decided to leave South Africa. We came to the United States to look for jobs. I was recruited by Vanderbilt University, and a year later I came here.

What do you think about recent events in South Africa?

Recent events have proven me totally and completely wrong. I believed in '82 that there was absolutely no sign of peaceful change, that if anything could happen it would be through violent revolution. The idea of change through the ballot box was inconceivable, and when I say violent revolution I'm talking about the kind of civil war where millions die. This was to me the only outcome. I'm absolutely delighted with the change that happened - peacefully, with goodwill on all sides, and that the oppressed people have not in any way tried to take revenge. So, had I been able to read the future, had I known twelve years ago what was going to happen, I probably would never have left South Africa. I'd have stayed and done my part in the society in which I had been brought up. As I say, I'm totally thrilled with what has happened, but I'm here now and it's going to be a bit difficult in this stage of my life to go back. I think I'm going to be here probably the rest of my days.

Can you tell me about your educational background?

Certainly. I grew up in Cape Town ... so I went to the University of Cape Town. I got my first degree there, a triple major in mathematics, applied mathematics, and mathematical statistics. When that degree was finished, my mentor, Walter Schaffer from physics, suggested that I move into theoretical physics. So I did a one year Honors degree in theoretical physics ... then I started an MS/Ph.D. program in theoretical physics, and after one year I was selected for a leadership training program in Israel over the summer break. I went there and visited the Weizmann Institute of Science which is one of the world's top research institutions in theoretical physics.... I met some people, spoke to some professors, and the next thing I knew I was registered for an MS in theoretical physics. However, there I discovered, working with the best in the world, that I wasn't in the top ten in theoretical physics; I wasn't in the top twenty, and in fact I probably wasn't in the top million, and that perhaps theoretical physics was a mistake. I went back to South Africa and did a Ph.D. in applied mathematics, where I really started off, and took a position as a professor in

applied mathematics. For various reasons, I was not really happy in applied mathematics. It didn't seem to be the area for me. So, in 1976 I applied for a job in CS. What I found funny was that there were twenty applicants for the job, and I was the only one who did not have one of the following: either a Ph.D. in CS, teaching experience in CS, or research experience in CS.... I got the job and I was, to my mind, the least qualified, and at that point my whole career took off. I was rapidly promoted from lecturer to senior lecturer to associate professor over a very short time. My publications went in leaps and bounds: from two I went to thirty in a few years. I spent a sabbatical leave back at the Weizmann Institute and that had a remarkable effect on my career. I met an American scientist, UCLA professor Dan Berry, also on sabbatical there, who gave a course in software engineering which was then an emerging field - I'm talking about 1979. He gave this course and not only did I get an immense amount out of the course, but I was also totally turned on to the whole area of software engineering, and that's been my area ever since.

You mentioned your dismay at not being in the top ten in theoretical physics. Is that important to you?

Yes, it is. It seems to me that if you're working in a field and you're just an also-ran, then you're not making any contribution to the field. To me, what is important is that my publications in SE are read, my SE textbooks are used, that my students, my Ph.D. graduates, get good positions. All my life, I've been an overachiever ... had a very strong drive for success. If you're in an area and not getting anywhere, I would find that frustrating. I found theoretical physics frustrating because I realized I'd never get anywhere in it. I left applied mathematics for the same reason.

What exactly do you like about computer science and software engineering?

That's a very very good question. What I like about CS and SE, what I've liked about everything else I've worked on - is that there are real challenges. Nothing for me in this area comes easily but that applies to all the areas I've worked in. There is a further dimension. SE in particular - and I do consider myself a software engineer rather than computer scientist - for me it is practical field. It is being used by real people in their real lives. When things go wrong with software not only can it irritate people, but faults in the wrong kind of software can kill. On the other hand when software goes well it makes everybody's life easier. There's software everywhere, in washing machines, car carburetors, watches, airplanes, nuclear reactors and

everything else. Computers are all-pervasive these days. That's the hardware though, and it's useless without software. So software's got to be all-pervasive. It's important that software be used to help humanity, add to civilization, and not to detract from it. When I was working in graph theoretic algorithms, although it is conceivable that something I developed could have helped humanity ... that was extremely unlikely. However, I know that people are using my methods in industry, and that is the reason I'm so happy in SE and get such a sense of satisfaction. Here is a challenge and I like meeting challenges. I believe I can contribute to society in a more direct way than someone who develops an algorithm [whose] effects may be twenty-five or fifty years down the road or maybe not. In SE there is this much more immediate response.

What areas of SE are you investigating?

We had major breakthroughs in the 1970's with the structured paradigm. Now we know that wasn't a major breakthrough at all. That paradigm worked for small-scale software. For large-scale software it didn't do the trick. The answer today seems to be the object-oriented paradigm. I'm looking at the following: One, I believe in the object-oriented paradigm. I really do. I've got a gut feeling that this is a silver bullet. I wouldn't be so presumptuous as to say ``the silver bullet.'' But, the object-oriented paradigm as a silver bullet is going to get us places. Am I right? I really want to know. I have come up with a lot of theoretical reasons in my writing why it is the way to go, read a lot of reports of successful projects with the object-oriented paradigm, but people don't seem to publish unsuccessful reports. If you fail, you don't want to tell the whole world about it. One of the challenges is: is the object-oriented paradigm a silver bullet or not? Since 1979, my number one objective has been to develop what I call the unified theory of software production. The term comes from theoretical physics, Einstein's theory....

But he failed to get it, right?

Well, he came up with it, but we don't know whether it's correct or not. It's in such a form that it's unprovable. Talking as a software engineer, he failed to get it. Speaking as a mathematician, he got it. But, certainly I want a unified theory of SE. That is to say, is there a unified approach, starting from requirements and ending up with maintenance and finally retirement, that one can use. And yes, I've come up with that sort of thing. The previous version of my major textbook is called ``Software Engineering.'' The new third edition is called ``Classical and Object-Oriented Software Engineering,'' and I think the title of the book says a lot about my outlook in life. First

of all, we can't throw away all classical SE and say, ``Heck, that's what we did in the past, and here's the future.'' We've got a large amount of software sitting there. We can't abandon it. Second, we can't simply say, ``Object-oriented is the way to go and let's forget everything we've ever learned about classical SE and do things in a new way.'' That's not the way the world behaves. People move slowly. Technology transfer is not instantaneous but a slow, steady transition. We have to move from the structured to the object oriented. This is something which interests me, the way people can move across this. The other challenge I'm working on is coming up with a complete unified theory of object-oriented SE. It's premature in that we don't know enough about it. Data, results of using the paradigm, and the drawbacks are coming in now. We are discovering that not all is rosy in the garden. I see [the unified theory] as a very long term goal. I'd like to able to report in ten years time that I've found something. In five years time, I doubt if I'll come up with a complete theory. That's the other reason why my book covers classical and object-oriented [methods]. Parts of it are still classical where we don't have the object-oriented answer. The best we can do is to use old methods, till we can come up with something new. The way I see software developed today is very much a hybrid method. We're using some of the old techniques, where we don't know better, together with new techniques, where we do know better, and we are hoping to throw out the old, bring in the new, and unify so there's some overall process and at the end everything will be object-oriented to a large extent.

Given your background, when you say ``Classical and object-oriented SE,'' it sounds, to me, similar to ``Classical and Quantum Physics.'' Is that deliberate?

It wasn't deliberate, but probably my subconscious came up with it. I really liked your analogy for the following reasons: First, classical physics, let's take Newton's law of gravitation, it seems to be correct. But, under certain circumstances, namely high velocity, Newtonian mechanics is not accurate. You need to correct it using Einstein's. Now, if you're going to write a hundred line program you don't need object-oriented anything. With 5000 or 50,000 lines of code - I know one shouldn't measure programs in lines of code but just to give some sort of idea - the structured method is working sort of okay, just as Newtonian mechanics works with aircraft or whatever. Once we start going into space, we need Einstein's corrections, the same way once we start dealing with larger projects we need the object-oriented paradigm. I see the object-oriented paradigm in some ways as a completely different thing. Some people say

quantum mechanics is totally different from classical mechanics. I would agree with that in general. Another way of looking at it is that at low speeds the two are equivalent. Elsewhere, one is a modification of the other. I see myself as an evolutionist, not a revolutionist. I would be hard pressed to talk about the object oriented revolution - it's evolution. The reason is that there are people out there who have been doing things the classical way so long and with relative success, that I don't think we should throw the baby out with the bath water.

What happens after objects?

Ah, now that's a very good question. I've seen papers on things that come beyond objects. Quite frankly, I don't believe any of them, but then when I first saw objects I didn't believe them either. I take some convincing. I'm essentially a pragmatist, and if theory doesn't lead to improvements in SE practice, I'm skeptical. In the case of objects, I read the theory, and it seemed to lead places, but I wanted to be more careful. So, in my earlier books I simply said, ``There are ways of doing it and object-oriented is one of them.'' The latest book says, ``The best way to do it is object-oriented, but if something has already been written in an old-fashioned way, you better continue maintaining it that way.'' It's conceivable that some of the things that have been put forward will be the next thing, or maybe they will not. But, one thing I can tell you with absolutely one hundred percent certainty is that objects are not the end. We will go beyond objects. I'm not usually scared about sticking my neck out, but in this case I'm not going to say anything simply because the whole object-oriented paradigm is just too new to be able to see what lies beyond on the other side of the hill.

What will software look like in thirty years? Will we recognize it?

I've got 20/20 hindsight, but foresight is more tricky! It will be totally unrecognizable. I'll give you a couple of reasons. First, my first programming language was machine code, that is to say, zeroes and ones. Now, we are way beyond that, and I think if I were to show machine code to people who code in C++, they would not recognize it. They would say, ``What is this? It looks like a core dump.'' ... In 29 years, things have changed dramatically. In another thirty years, let me make a prediction I think is very safe, there will not be programmers. I don't think there are even going to be programmers in ten years time, because what we've got now is code generators. In terms of levels of abstraction, we were working in 1965 at the machine level, zeroes and ones, now we are at some high-level language. With code generators you are

working at the level of detailed design. Now, you get from your specifications to detailed design, and you press a button, and out comes code probably in C, in some cases in COBOL. I say ``probably'' because you are not supposed to look at the code. The whole idea is that you use code generators and you never, ever look at the code... What I am willing to state categorically is that code generators are going to abolish the job of programmers. Their job will be replaced by that of the designer - someone who goes down to the level of detailed design, just as the job of machine-code programmer has now been replaced by high level programmer. The question is: what level of abstraction are we going to be at in thirty years? ... I'd like to say - I'd be lying if I said it - that all we have to do is write down specifications in some kind of natural language and an automatic program generator would take the specification written in English and would generate it for me. That I know will not happen in thirty years time, in 300 years time, perhaps, but not in thirty. What I hope to see in thirty years is that the level of abstraction at which we work will be higher than even the detailed design. That is as much as I can hope for in thirty years....

Regardless of the level of abstraction, specifications for instance, you'll still need someone highly trained, an expert, to do it. So, will software ever be easy?

Hmmm. Another tough question.... I'd like to say ``no'' in order to convince your readers that they have a career ahead of them, and I do believe that. I think software is going to divide up. There will be some software that will be easy. Some will have some kind of automated program generators to handle the humdrum stuff. I believe what's going to happen is that software engineers will split into two groups. The work of high level people will never be easy. All these tools I'm talking about will be developed by them. Then, there will be people using very advanced tools in order to develop standard software, very much as people who know nothing about programming nowadays use word processors, and people knowing even less about programming write very advanced programs using spreadsheets. So, I would say [using] word processors and spreadsheets is simple programming and the people who write the word processors and spreadsheets are very advanced programmers. Notice I don't use the phrase ``sophisticated programming.'' I learned that when I was is Israel as a student in 1969. I was invited to a cocktail party, and I met Ezer Weizman, who is now the president of Israel. He had been the head of the Israeli air force. I was talking to him about sophisticated aircraft and he very much took me to task. He said, ``There's no such thing as sophisticated aircraft. There are sophisticated pilots, but

the aircraft itself cannot be described as sophisticated. It may be technologically advanced, and have wonderful features, but the word sophisticated is applied to a human being, not to aircraft....''

I would not talk about sophisticated tools in the future. Some people will be sophisticated users, just as there are some sophisticated users of spreadsheets doing some very advanced work using spreadsheets. It's not that the spreadsheet is sophisticated. Again, as I said we are going to be divided into two groups; some will write the spreadsheets and the other people will just use them.... I think the dichotomy between programmer and software engineer will become greater, and software engineers will be working on harder problems than we are working on at the moment. Programmers will have their life made much easier.

What about the dichotomy between user and the software engineer?

Hmmm. This is something that really frightens me. I am a bit scared of end-user programming. Software engineers are trained to mistrust everything that comes out of a computer. End users are trained to trust everything.... The moment you let an end user do his or her own programming the danger is that mistakes will be made and mistakes have been made. We know of companies that have gone bankrupt or nearly bankrupt as a result of end users deducing certain ``facts'' or corrupting databases. The solution perhaps lies in education.... No, we did not learn CS in high school. My son and daughter are in high school, both doing courses in computing which are required courses. OK, these courses are more in applications. That is to say, spreadsheets and word processors, and limited programming. But at least they are learning to program at high school. If they were to take a job with no further education in programming and simply were to do end-user programming, I think merely what they've learned in high school is enough to let them realize that the first time you write something the answer is not necessarily correct. Similarly, once programming becomes as pervasive in the schools as arithmetic now is, and I can see that coming a lot sooner that the thirty years of your previous question.... Once this happens people will be computer literate when they leave high school. One part of that is not just how to use a computer, but having a healthy skepticism regarding the output from a computer. The situation now is that if a clerk gives an answer, and you say, ``That doesn't make any sense,'' they say, ``It must be so, because the computer says so.'' I think that will go away slowly with education. Once that skepticism is in place, with the aid of these tools which I spoke about, we'll have more work done by non-programmers and maybe

the job of programmer will go away completely. Perhaps we'll just have two job categories - software engineer, whatever that means in twenty years time, and then someone who uses and programs a computer. Some kids in high school are using hand-held calculators, some of which are very advanced with graphics capabilities. Those kids don't call themselves mathematicians. By the same token, there will be people who use small hand-held computing devices, and they won't call themselves programmers any more than I call myself an arithmetician when I work out whether the change I'm given by a cashier is correct.