Intersect, Vol 6, No 2 (2013)

"Computer Science and X": An Interview with Steve Cooper

Andréa Slobodien
Stanford University

Steve Cooper is an Associate Professor (Teaching) in the Computer Science Department at Stanford University. He earned his PhD and MS in Computer Science from Syracuse University and his BA in Mathematics and Chemistry from Cornell University. His research areas lie in program visualization and semantics. Together with Randy Pausch and Wanda Dann, he developed Alice, a freeware object-oriented educational programming language aimed at appealing to populations not normally exposed to computer programming. From 2007-2009, Dr. Cooper worked in the National Science Foundation's Division of Undergraduate Education, within its Education and Human Resources Directorate. There, he worked as a program manager, and worked on the CCLI, ATE, NSDL, and S-STEM programs. He also serves as Chairman of the Board of Directors for the Computer Science Teachers Association.

Computer Science and Public Policy

ADS: What brought you to Stanford University?

SC: I came to Stanford in Fall of 2010. When they made me an offer I was really excited. I had never been on the West Coast before; I had never taught at a very elite school before; I had never been at a school where there is such a devotion and dedication to teaching undergraduate computer science, which is really the thing that I found most exciting. It was a really exciting opportunity to get to work with faculty like Mehran Sahami, Eric Roberts, Julie Zelenski, and Nick Parlante who I had known from their reputations as leaders in the CS education community.

ADS: You teach a course on computers, ethics, and public policy (CS 181), and you also feature some interesting STS conversations at the end of your introductory computing course (CS 105). What do you see as your role in the intersection of computer science and areas like policy, education, and ethics?

SC: The funny part is that when I came to Stanford, I had never taught an ethics course before, and when the department asked me if I'd be interested in doing it, I spent an inordinate amount of time preparing. I think that first year I must have read four books a week just to try to get myself up to speed. When I was at the National Science Foundation (NSF), I had been aware that there was a huge disconnect between the technical side of the field and the policy side. I remember one Christmas when everyone from NSF was gone and I was asked to draft the NSF policy piece for a response to some possible legislation in Congress regarding cybersecurity education. I remember thinking, *Really? Don't you have anybody better to do this?* I was told, "Even during the year we wouldn't have had anyone *better* to do it. Someone would just have to do it." And it got me thinking: Public policy is so important in terms of dictating the directions of what we can and can't do and where our field is going to go, and NSF is such a central place for science, yet there aren't people who are really experienced in the technical *and* policy side. That's a problem. When I came here, I was very excited to get a chance to teach that class and to make it so that more people would be able to recognize the importance of connecting technology to policy. There are very few schools in the country that even offer an STS-like major. Here, the challenge, of course, is that the Silicon Valley has far more jobs for someone who's an STS major, so we can't even send STS majors to Washington, because there are no extra students. It became a chance for me to get CS, STS, Symbolic Systems (SymSys), Math and Computational Science (MCS), and engineering majors thinking critically about what's going on when you mix technology with policy. Certainly, a couple of years ago when the *Stop Online Piracy Act* (SOPA) and

PROTECT IP Act (PIPA) almost passed, Silicon Valley didn't respond in a timely manner. There's something really wrong when we are driven by legislation by people who really don't understand the pressing issues that we need to address.

ADS: How do you feel about the response to SOPA and PIPA, with major websites threatening to shut down?

SC: So that was kind of late. Silicon Valley companies could have been more clever about this and they could have launched public action campaigns, like Mark Zuckerberg is doing vis-à-vis immigration legislation. If you look at the reaction to the *Cyber Intelligence Sharing and Protection Act* (CISPA), which happened a couple years later, we see that the Silicon Valley hasn't learned. They didn't get involved in public policy, so we got CISPA back again, which is fine for some companies but not for most.

ADS: Just as there are programmers in the Silicon Valley who don't understand policy, do you think that there is also a lack of engineering knowledge in DC? For example, do you think there are young policy interns who don't know enough about programming but who are bound to become our next leaders?

SC: Without a doubt that's the case. When I was teaching on the East Coast, I remember going to visit my senators from the state of Pennsylvania. I remember meeting Senator Rick Santorum's aide in response to higher education and technology, and she was a lovely English major fresh out of whatever Ivy League program she had come from, but she didn't understand computer science. Even when I met Senator Arlen Specter's aide, she was not technologically savvy. The awareness is not there. When you look at Capitol Hill, it tends to be mostly humanities majors who get thrust into tech because they're the most junior person on the team—not because they really have a computer science background. I'm not sure why that is. Perhaps it doesn't pay well; perhaps it's not glamorous. Certainly, if you look at the members of Congress, there are no computer scientists. I don't even think there are any engineers. Out of 435 members, there's not going to be a tech presence at all.

ADS: In CS 105, which is an introductory programming course, you feature a lecture at the end of the quarter about current issues in computer science. What is your goal when you are doing that?

SC: CS 105 is largely comprised of students who take it as an exploring step, but many are trying to finish their engineering requirement. When I do that lecture at the end, I try and open a sense of awareness. For most people, it is the only computer science class they're going to take. These

issues are there, and they're going to be more important in their lives than ever. Who knows what one lecture can do, but maybe one lecture can convince a student to go on and take a couple more computer science classes—not necessarily to be a computer science major, but to have enough programming skills to be a technology policy advocate and to have a comfort with technology. Certainly, in Silicon Valley, many humanities majors have a comfort with technology, so maybe I can help to develop a much better person to have in Washington than many of the folks who are already there.

Computer Science and Ethics

ADS: What do you see as the responsibilities of programmers? To hack away and create beautiful, elegant code? Is there a greater responsibility?

SC: Definitely the latter. What I try to do in the CS 181 course is show that technology isn't value-neutral. As a result of what you build, you're pushing a particular set of values. There are a couple of recent articles that talked about how the Silicon Valley is determining privacy policy for the nation. In building the technology to do or not do certain things, we are de facto making the case for and against personal privacy. Computer scientists can't say, *I was just working there, I was just making a program, it's not my fault*. It is your fault if you build something that's inappropriate, and you have to be aware, and you have to spend time thinking about the issues involved in what is being built. It's something that software engineers aren't preparing themselves to think about, but they must think about it. We are fortunate that so many students take at least one computing, ethics, and public policy course so they have a chance to see this. It's not just about the chance to make money by building code. What you're choosing to do really matters.

ADS: What literature and film do you consider mandatory for every programmer?

SC: Good question. I'm going to be somewhat biased because I'm a product of a liberal arts education, and I think that is important. I'm in a computer science department that is housed in an engineering school that typically doesn't assign the sheer volume of reading and literature everyone should know. I'm clearly biased in favor of being extraordinarily well-read in English, Anthropology, History, Sociology, etc. One of my favorite Turing Award winners is Alan Kay, and he has published his reading list. His reading list doesn't have a whole bunch of computer science books. It has a whole bunch of books in the social sciences and humanities. I recognize that because we are in an engineering school, I probably can't get every computer science major to read my list of the top 100 books, but it doesn't mean I wouldn't like that as a background. It

certainly is my own influence, coming from an undergraduate program as a science major in a school of arts and sciences. I did a year of Economics, Philosophy, two years of English, and a year of American Government, so that background was very helpful in terms of making me who I am. I think that is important for students, though it's harder in a school of engineering where students simply don't take that volume of classes.

Computer Science and Education

ADS: Do you think that it's better to develop computer scientists into better writers and communicators, or is it better to convert non-technical majors into people who understand code?

SC: I'd answer both. When you sit in a classroom of 170 students, it's very hard to know which are the students who are going to get the opportunities to be called to effect policy. Which are the students who are going to be the ones before congressional subcommittees? I don't know that. I don't know who's going to be ready, but I'm certain that in that classroom we are going to have a bunch of those people influencing public policy sometime in the not-too-distant future. I kind of duck your question by saying both because I think both are needed and I don't know which is the best.

ADS: Stanford tries to make us take introductory humanities courses, and I still remember reading The Republic. Plato's Form theory is how I came to understand object-oriented programming and instance variables. Do you think that those kinds of connections are really important for programmers?

SC: I do. I think a Platonic view of an object versus, for example, an Aristotelian view of an object, actually helps give you a perspective on object-oriented programming: does the mountain have beauty because it's there or is it from your interactions with it that the mountain is beautiful? We can consider the concept of the meaning of an object in object-oriented programming in much the same way. I think those things are very helpful for students, and I think that's part of the well-roundedness of an education. I would like to see students having a strong humanities background because I agree with you. I think you can get a great deal out of reading ancient philosophy. I think that's an incredibly valuable thing to do, and it gives you a very different perspective on your own science.

ADS: Computer science has this rare quality of being elite and accessible at the same time. On the one hand, if you grow up with a computer, no matter how good that computer is, you can teach yourself how to program. On the other hand, programming is so complex that if you're not a genius, you need to go to a university to learn these skills. Do you think that

there's something that could be happening in education that would help us create more programmers?

SC: This is a real challenge. When you look at the formal K-12 school system in the US, computer science doesn't play a huge part. On one hand, that's very sad, because many kids don't get exposed to computing. On the other hand, maybe that's not such a bad thing because there simply aren't enough super K-12 teachers yet and we have to figure out how to address that. Maybe it's a chicken and egg problem and maybe the schools of education have to produce more. There are teachers in K-12 who are wonderful computing teachers, but there are nowhere near enough of them, and to produce enough would be an enormous national undertaking. Having kids get a positive exposure to computing is important, and we're exploring how to do it. We are exploring through informal means: I'm running a summer camp for kids who have no previous computing experience. It's only going to impact a small number of kids, but it's going to be a number of kids nonetheless. I run a workshop for teachers to try and develop that expertise, because it's hard to lobby for more exposure in K-12 if we are not going to be able to get teachers. If you look at the average starting salary of teachers and the average starting salary of a Google software engineer, the difference is massive. How are we going to get some of the best people in computer science to become high school teachers or middle school teachers if the pay is one-third or one-fourth the rate and the benefits are worse? Of course, I do advocate very strongly for exposure in K-12, but I worry about the capacity to do that. Where are we going to get the teachers and how are we going to produce the teachers? Lots of people are trying online approaches. There are a bunch of programs to learn computer science on your, but I think the jury is still out on the success of being able to teach yourself computer science. I think it's hard to do and it's very hard when you get stuck, because it's a meta-level. It's not using an algorithm, it's writing the algorithm. It's hard to get the right kind of help that you need, so I worry about whether these pure online solutions are going to help enough kids. Kids come to Stanford, and many become CS majors who have never had a CS background before college, and I think that's a good thing. I think colleges having these absolutely superb CS1 classes is an absolutely essential thing. Many kids come to Stanford and say, *I didn't realize I was good at that, and I had such a good experience in CS 106A. Wow I'm going to become an STS, SymSys, CS, MCS, etc., major, because I'm actually good at this type of problem-solving*. I'd love to do it in high school, but I'd also like to push the colleges toward trying to create outstanding CS1 classes, and too few of us are doing so.

Computer Science and the Silicon Valley

ADS: Who do you think are the most influential people in computer science today? Are they all programmers?

SC: If you look at the most influential people, you look at people like Don Knuth, who is a retired computer science professor here. When you look at the influence he's had on our discipline, it's absolutely stunning. Computer science doesn't have a Nobel Prize because we weren't around at the time of Nobel. We have the Turing Award. If you look at the Turing Award winners, those are the giants of our field, because those are the ones who are influencing the direction of studies 10, 15, and 30 years from now. They're not the ones who are in the news. They're not the Bill Gates, Mark Zuckerbergs, or Sergey Brins of the world, because the people I'm telling you about aren't going to be the richest. When you look at the ones who influence the discipline, it's likely going to be the ones who are researchers and academics—not the ones who become billionaires with a company. Sure, one can argue that Microsoft could have a great deal of influence over anything they want because of sheer volume, the number of people they hire, and the money that they have. Perhaps they can and do have influence.

ADS: It seems like these Mark Zuckerberg types are entrepreneurs who knew how to code, but they blurred the line between entrepreneurship and engineering. What do you think about the divide between computer scientists who do research in the Gates Computer Science building all day and the ones who hack something together and focus all their time on selling it and making profits?

SC: You need both of them. You need the people pushing the field further, and you also need the people producing the products. Stanford tends to represent both quite nicely. Clearly, being in the Silicon Valley with the entrepreneurship culture and all the venture capital funds available, it's a really exciting place to be, and you can do both. The funny part is that you don't even have to do both individually. You have PhD students who go off to found companies. One of the co-founders of VMware just came back to finish his PhD 18 years later. You can have your cake and eat it too. You can move back between one and the other very smoothly if you wish to, and that's a nice thing, and both are needed from a practical perspective, because you certainly don't want the field to stagnate. But you also need people to produce products because we are living in the information revolution. We are the best ones at handling information and trying to make sense of it.

ADS: The number of CS and STS majors at Stanford is booming, and CS is now the largest major. Why do you think that technical majors have become so popular? Is it because we are in the Silicon Valley, or is it a response to something greater?

SC: On one hand, CS is practically the only field that wasn't hit by the recession. Parents have been pushing their kids toward something that is career-oriented, and computer science has been fairly immune from the economic downturn. I've seen times where people have gravitated to CS for purely economic reasons, but this isn't necessarily one of those times. There's a certain sense that the need for CS in your field, whatever that field may be, has grown and become more obvious. A lot more of the students taking *Programming Methodology* (CS 106A, or CS1), *Programming Abstractions* (CS 106B, or CS2), and *Programming Paradigms* (CS 107) aren't CS majors and have no desire to become CS majors but end up needing it for their field. We have seen a huge growth of CS majors, and I think the economy is part of it. I think the popularity does tend to ebb and flow a fair bit. My guess is that interest will wane some time in the future, but the need for computing isn't going to go away. Students who are studying Economics are going to need to be taking CS classes whether or not they want to. Their discipline is going to dictate it, and that's a change that we will probably never undo. Man, it's a popular time to be CS in the Silicon Valley, that's for sure!

ADS: Do you think that universities are graduating the amount of software engineers needed, or is there a chance that this bubble will burst and there will be less of a need for CS majors in the near future?

SC: My colleague, Eric Roberts, puts together job projections. We are producing less than one-third of the number of projected job openings according to the Bureau of Labor Statistics. The U.S. is producing just under 40,000 CS graduates a year.[1] This is a little under one-third of what industry says they need. Part of the big push in immigration reform is because industry says they need labor, and we aren't producing anywhere near the quantity of labor needed. While I expect certain industries to bubble, I don't expect the need for computer scientists to go away any time in the near future because there's such a disconnect between the number of available computer scientists or CS-related majors and the number of jobs. Companies have been trying to outsource for years and years and we don't know how to do that. We figured that we would go to Eastern Europe or Southeast Asia, but we haven't figured out how to effectively outsource any but the most menial of jobs. Industry is, to a certain extent, stuck. They're fighting like mad to bring more workers here because jobs simply aren't getting filled. Just go to the CS department job fair and look at the buzz inside that massive tent. They're trying to hire 15 times the number of people who are in the tent. The disconnect between labor and need for labor is real, and I don't see that righting itself in the near future.

[1] Information from https://webcaspar.nsf.gov/

Computer Science and Gender

ADS: With the increase in CS majors at universities, what is happening with the gender imbalance? Do you think our department is doing innovative things in this regard?

Gender is a huge issue. It's an issue with all under-represented groups, but I'll try and talk specifically about gender. The year before I got to Stanford, three out of 69 graduating students were female. Now, about one-fourth of our graduating CS majors are female. That's not enough. It should be 50%. We are getting there. I think Stanford has built up a core group of very excited and energetic young women who are excited to lead the charge, because it's very hard for a bunch of male professors to lead this. We can encourage, push, help, and work with the students to be able to run wonderful programs. A few times a quarter, we host a *Women in CS* dinner, and we've run out of big enough rooms in Gates to host all attendees. It helps that it's driven by the students, rather than from the top down. And it helps that this group is encouraging the younger generation of women to become the next leaders. It helps build leadership experience and it's a great way to help address this terrible imbalance. Nationwide, it's an embarrassment, and it's still not where it needs to be at Stanford. We are getting better, but we still need 50% of our majors being women, not 25%. We need to look at what other schools are doing successfully. We invite speakers in from other schools and ask what they're doing that works, and we steal their ideas. She++ recently put together a documentary, and you hear speakers talking about the fact that this is a national issue. In fact, much of the disconnect between available workers and available jobs would be solved if the percentage of women became equal to the percentage of men who are in CS. Some campaigns are probably more successful than others, but we just have to keep trying until we get there.

ADS: Why is a gender balance so important?

SC: As soon as you don't have enough workers of *any* demographic working in CS, you don't have the maximum number of workers. It's a labor issue. It also becomes an issue because the more eyes you have looking at a piece of code and the more hands you have producing a piece of code, the better it's going to be and the more varied perspectives you'll have. You're going to get better quality code. Do women program differently than men? Maybe they do, maybe they don't. But it's a work force issue. By simply telling women, *no you are not going to be computer scientists*, we are not meeting the labor need. Computer science jobs also tend to be very high-paying jobs. They are jobs that have aspects that tend to be amenable to telecommuting and building families. When you look at some of the companies in the Silicon Valley, there's support of

daycare facilities for kids, and it's really easy to be involved in a family. So why do we want to be sending women away from these sorts of jobs rather than sending them towards these jobs?

Computer Science and the Future

ADS: Object-oriented programming is a new concept compared to the kinds of computer science that we were studying years ago. How do you think the major has changed since first coming to Stanford, and what changes might we expect to see in the future?

SC: One of the things we tend to forget is that our field is extraordinarily young. The term "software engineering" only dates from 1968. So if you were alive in 1967, people did build systems but software engineering didn't exist as a term. In trying to answer your question, there are a couple of aspects. The technology is changing incredibly fast, so we have to make sure that the students are able to change with the technology and the times. It's not a matter of learning some language or some technology, because I can assure you that in 5 years, the popular language is going to be different. I don't know what it will be, maybe it will be Python, but it's going to be a language that wasn't widely taught a few years ago. The most popular language 5 years after that probably hasn't been invented yet. We have to get students ready for the technology, and the basic problems we have been facing as a discipline for the last 20-40 years haven't changed all that much. They keep getting rediscovered because they've been hard problems, and I expect that to continue. Security is going to be big, and parallel and concurrent programming is going to be very big. With the breakdown of Moore's law, the solution is to put more CPUs on the chip, but you're going to have to learn how to be able to exploit that. Though we were struggling with concurrent and parallel programming in 1970, we are still struggling with it. The next generation is going to discover that it's still hard to do. It's still hard to build reliable systems, and security still matters. Handling big data is going to be another one, though we've been struggling with big data forever, the difference being that now we've got better techniques to deal with it. I see all of these fields as growing. I don't know if they're new problems, but they're new views of existing problems that we haven't done a good job of solving. One of the most important growing areas is the intersection of computer science with x: whether it's with finance, photography, sociology, English, or literature, virtually every major is finding itself needing to interact with computing, and I would expect to see the growth of many of these new fields. This produces something new because sociologists, for example, have been worried about certain questions for centuries, and computer scientists didn't even know these questions existed, much less how to solve them through computing.