Interview with Margo Seltzer

RIK FARROW

Margo Seltzer is Canada 150 Research Chair in Computer Systems and the Cheriton Family Chair in Computer Science at the University of British Columbia. Dr. Seltzer was also a co-founder and CTO of Sleepycat Software, the makers of Berkeley DB. Her research interests are in systems, construed broadly: provenance systems, file systems, databases, transaction processing systems, storage and analysis of graph-structured data, synthesizing system software, discrete optimization, and applying technology to problems in healthcare. She serves on the Computer Science and Telecommunications Board (CSTB) of the (US) National Academies and the Advisory Council for the Canadian COVID-19 contact tracing app. She is a past President of the USENIX Association and served as the USENIX representative to the Computing Research Association Board of Directors. She is a member of the National Academy of Engineering, a Sloan Foundation Fellow in Computer Science, an ACM Fellow, and a Bunting Fellow. She is recognized as an outstanding teacher and mentor, having received the Phi Beta Kappa teaching award in 1996, the Abramson Teaching Award in 1999, the Capers and Marion McDonald Award for Excellence in Mentoring and Advising in 2010, and the CRA-E Undergraduate Research Mentoring Award in 2017. Professor Seltzer received an AB degree in applied mathematics from Harvard/Radcliffe College and a PhD in computer science from the University of California, Berkeley.

Rik is the editor of *;login:*.
rik@usenix.org

I first noticed Margo Seltzer because she had brought her baby to a USENIX conference in the late '90s. That was unusual, as I had seen few parents with their children at conferences. Later on, I got to know Margo better when she was on the USENIX Board and I was routinely attending board meetings.

What prompted me to ask someone as busy as Margo for an interview was her keynote address at the 2020 USENIX Annual Technical Conference entitled "The Fine Line between Bold and Fringe Lunatic" [1]. I recommend watching Margo's talk, but the gist is simply this: you are likely to have a more interesting career if you are willing to take risks. That's not what Margo actually says, just my own interpretation. She wants researchers to broaden the subject areas they keep abreast of as well as to consider researching at the frontier of knowledge.

Rik Farrow: You became a faculty member at Harvard, working in CS. Was that at all unusual?

Margo Seltzer: I think what was unusual was that I turned down a position at MIT (arguably ranked #1) for Harvard (pretty much unranked except in theory where we had Turing award winners).

RF: I don't understand why ranking is important. Could you explain what the ranking means to someone taking an academic position for those of us who won't have that experience?

MS: Ranking's importance varies by who you ask.

There has been a lot of data analysis about the network formed by studying the migration of PhD students to faculty positions. New faculty typically have degrees from institutions from rankings higher than the ranking of the school in which they are teaching. So if you want to teach at a top N school; you'd better get a degree from a top (N-1) school. And if you want your students to get jobs at a top N school, then you want to be teaching at a top-1 school (or at least one of the "big 5").

Unfortunately, rankings are a fuzzy metric—I advise undergrads going to grad school to place far more emphasis on the person/group with whom they will work than the ranking of the school, but students don't always listen. And students from undergraduate institutions without a lot of advising don't have much to go on other than the rankings: they don't know the faculty.

So the ranking of the university at which you take a faculty position is directly correlated with the quality of students you get and the likelihood of placing them at other top institutions.

Thus—turning down MIT (arguably #1) for Harvard (top N > 20 and probably closer to 30–40 then) was shocking to most. I was definitely called an idiot by some.

Harvard, in particular, had a dismal reputation for granting tenure. There had been a famous case in 1983 where a person widely regarded as a superstar in his community was denied tenure by Harvard. So when I got there (1993), Harvard had not tenured anyone in computer science since 1981. My colleague, Stuart Shieber, broke that curse by getting tenure in 1996. Then Mike Smith and I both got tenure in 2000. Since then, Harvard has done very well by hiring strong people and making sure they get tenure.

RF: You begin your talk [1] by demonstrating how computer science has been partitioned over the years, beginning with the split between hardware and software, then software splitting into operating systems and programming languages, and so on. You encourage people to cross the many boundaries that exist today, and I do sometimes see that happening, for example, with file systems using key-value stores for metadata. Do you have other examples?

MS: The crossover between file systems and databases has grown a fair bit over the past 30 years, but it takes a lot of pushing:

Journaling (logging) was developed in the database community to provide transaction support in the '70s. It wasn't fully embraced by the file-system community until the '90s or later. At this point, it's fairly standard.

Transactions are another concept with a history in databases—we now see transactions in hardware (i.e., transactional memory) and every once in a while in file systems.

Program analysis (e.g., static analysis, symbolic execution) grew out of the programming languages community, but it has been and is being adopted in systems for bug finding.

The emergence of persistent memory (e.g., Intel Optane) brings together work from systems (virtual memory and single level store), databases (persistent objects), and file systems (persistent files).

So these things happen, but a lot of the time the researchers themselves don't think to look at work of other communities and will re-invent the wheel instead of borrowing it and making it work better.

RF: I agree that not looking at what has been done in other communities really slows down research and innovation in CS. But isn't there an issue with the amount of research, just in the small niches that we have today, being too overwhelming for most graduate students to cover?

MS: It is impossible to keep up with all the work being done in a single field, so how can one hope to know what's happening in other fields? In machine learning alone, something like 100 new papers show up on arXiv every day. So what is an overworked graduate student to do?

It's not necessary to read every paper published to know what's happening in a field. The key is really an openness to what's happening in other areas, a curiosity, and a willingness to do the hard work of trying to understand work from a different community when it's appropriate. One of the first things I do is encourage new graduate students to subscribe to The Morning Paper—https://blog.acolyer.org/. My understanding is that Adrian Colyer, the author, is not really a computer scientist, but every week he sits down and reads about three computer science papers and writes up great blog posts about them. And he moves from area to area, reading whatever is recent or what is particularly interesting to him. I love his posts—I have a mailbox full of ones I've not yet had time to read.

Just reading Adrian's blog posts will give a student a broad introduction to a lot of areas. But even that isn't enough.

You have to be willing to talk to other people—not just the people in your lab but people in other labs. Go to weekly grad student social events and really try to understand what people are working on. Here is the secret: you are going to have to be willing to ask naive questions. I call them stupid questions, but they aren't really stupid, they are mostly just the questions that someone unfamiliar with an area will ask. And even more important (and possibly scarier), you have to be willing to say, "Um, I didn't really understand that, can we go even more slowly?" I collaborate with many folks who are way more mathematically sophisticated than I am, and I tend to ask (a lot), "Could you explain that to me in small words?" To be honest, it took me a long time to get over the knee-jerk reaction of just nodding and pretending that I understood what was going on when I was lost, but I learn a ton more when I'm willing to take that risk. And who better to take that risk with than your peers? And you never know, you might find an area that intrigues you, a topic of mutual interest, or just something new and interesting.

The key is not to be an expert in everything but to have a vague sense of what people are working on in other fields, so that when the opportunity arises, you can draw ideas from disparate areas and know what the areas are and perhaps even with whom to consult (that fellow student you were chatting with just the other day).

Super secret #2: being able to talk to people in other areas will be your single greatest superpower on the interview trail, where you're expected to be able to have intelligent conversations with people from different areas.

Fun story: In my interview that wasn't really an interview (or perhaps it was the non-interview that really was an interview) at Harvard, I was taken to lunch by two theoreticians—one Turing Award winner and one future Turing Award winner. They peppered me with questions to the point that I was still working on my salad when they got to dessert! But clearly something worked—shortly after I arrived, one of them dropped by my office to ask for my "expertise" on a topic...I was floored. What on earth did I have to offer a world-renowned theoretician? Well, he had some interesting ideas about applying his latest work to storage, and well, he figured that perhaps there might be people who knew more about storage than he did. It was a good lesson for me—no one is above asking questions, and no one should limit themselves to a small box, even if they are the absolute best in that box!

RF: Provenance is the first of the fringe lunatic ideas you cover in your USENIX ATC '20 keynote. The quest for provenance began early in this century, largely as a way to be able to recreate data based on its provenance: what had happened to that data since it was created. I recall thinking at the time that this seemed like a reasonable thing to do, but later wondered if having to maintain orders of magnitude more data as provenance really made sense. The story you told about provenance includes different groups taking different approaches, along with attempts to unify some elements. Were you surprised at where researchers had taken the original idea after almost 15 years had passed?

MS: Yes and no.

Around 2014, I "gave up" on provenance—I felt that the community was focused so much on provenance collection that they were not giving ample thought to motivating users to collect provenance. I was frustrated and basically went in other directions—then, almost immediately, I got two provenance proposals funded with collaborators. In one, we focused on use from the beginning—in the other, I forgot my own lesson for several years and only rediscovered it a few years later, fortunately with enough time to change course.

That said, all those are in the higher levels of the stack.

I am actually thrilled that the systems community has embraced provenance and is thinking hard about how to use it: security, information flow, reproducibility, etc. I always felt that system level provenance was the glue that could hold lots of things together, and these folks are making it work.

So am I surprised: 1) No—I don't think any of the things people are doing would have surprised me in 2006. 2) Yes—it seemed like the field wasn't going anywhere, but it still is!

RF: The other example in your keynote had to do with program synthesis, although to me it sounded much more ambitious than merely being able to generate a program. The DARPA BRASS [Building Resource Adaptive Software Systems] program was really about extracting intent from systems so that when circumstances changed, the system could adapt to the change and still succeed in accomplishing the intent of the system. You were among the "fringe lunatics" (your words) who took that to mean making the operating system adapt to new hardware by synthesizing operating systems from machine descriptions. That sounds like a ridiculously tall feat to accomplish, but a very good example for your theme. Could you tell us how that worked out?

MS: We didn't synthesize a complete system, but we've synthesized several parts of the Barrelfish operating system [2] and nearly an entire port of our OS/161 educational operating system [3]—and we've done this for about four different processors!

References

[1] M. Seltzer, "The Fine Line between Bold and Fringe Lunatic," 2020 USENIX Annual Technical Conference (USENIX ATC '20): https://www.usenix.org/conference/atc20/presentation/keynote-seltzer.

[2] The Barrelfish Operating System: http://www.barrelfish.org/index.html.

[3] D. A. Holland, A. T. Lim, M. I. Seltzer, "A New Instructional Operating System," in *Proceedings of the 2002 SIGCSE Conference* (February 2002), pp. 111–115: https://www.seltzer.com/assets/publications/A-New-Instructional-Operating-System.pdf.