Computing as a Discipline: Interview of Peter J. Denning

Interviewed by Sebastian Dziallas of University of Kent

January 29, 2015

This interview was part of a research project about ACM curriculum recommendations for computer science since 1968, and the corresponding shifts of perspectives about the nature of computer science. The full study was published as:

Dziallas, S. and Fincher, S. 2015. ACM Curriculum Reports: A Pedagogic Perspective. *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (New York, NY, USA, 2015), 81–89.

Interviewer: Let's begin with a broad question. I know you were responsible for the ACM/IEEE report "Computing as a Discipline" in 1988, but you had been active with education in ACM long before that, and you remained active after for example as Chair of the Ed Board. I wonder if you could talk a little about your perspective over the course of those 30 years.

Denning: Certainly. I've been involved with the ACM and curriculum ideas for a very long time. My first formal contact with ACM curriculum efforts came in 1970 in conversations with Sam Conte of Purdue, who was a member of the famous ACM Curriculum 68 committee. He was interested in having me come to Purdue and he finally recruited me to leave Princeton in 1972. He frequently sought my ideas and advice about computing curricula. Our conversations put the idea in my head that I could contribute to the computing curriculum in some way.

I was involved in curriculum design before meeting Sam Conte. In, 1969 Bruce Arden invited me to chair a task force for the NSF COSINE (computer science and engineering) project. They were trying to bring Computer science into engineering schools. I guess computer science was more accepted in a school of science or a school of mathematics at that time than it was in a school of engineering. The COSINE effort was intended to bring more computer science into engineering.

They asked me if I would chair a task force to develop a recommendation for a core course on operating systems. This was, at the time, a radical idea. The core of computer science at that time consisted of mostly mathematical courses that were applicable to computer science -- for example, automata, switching theory, discrete math, parsing theory, numerical analysis, and computability. But all matters involving programming, programming languages, compilers, systems, and software development were considered applications and were outside the core. Even compilers were outside the core. Although everyone saw compilers as a very mathematical subject, compiler courses were seen as an application of mathematics rather then the development of mathematics.

Operating systems was even further away from being a core subject. It looked like a hotch-potch of systems programming tricks, and it did not seem to have a set of core principles. But because of the success of the 1967 Symposium on Operating System Principles, organized by Jack Dennis and Walter Kosinski, the leaders of the COSINE committee were convinced that there were core principles in operating systems. They asked me to organize a task force to specify a course in operating system principles that would strike the academics of the day as worthy of being in the core curriculum.

Programming languages were another important topic that was at the time outside the core of computer science.

I put together a small task force of OS leaders -- Jack Dennis, Butler Lampson, Dennis Tsichritzis, Richard Muntz, and Nico Habermann. We quickly came up with a recommendation on a core course in operating systems. The COSINE committee accepted the recommendation and publicized it. Within very few years there were several textbooks on operating systems that used our report as the framework for their tables of contents. The first was by Nico Habermann, a member of the task force. A copy of the report is available from my website, http://denninginstitute.com/pjd/PUBS/cosine-8.pdf

Interviewer: You said COSINE was an NSF-funded project. What did they pay for? Was Habermann's book part of the project? Was there any other post-project work by members of the team?

Denning: Yes, the National Science Foundation supported the COSINE project. Most of the budget was travel expenses so that the task forces could meet and develop their recommendations. The project also paid for the printing and distribution of the task force reports.

Habermann's book was not part of the project; it was his personal effort after the task force was disbanded.

In 1972 I wrote a follow-on paper for the Spring Joint Computer Conference with a long title, "Operating System Principles and

Undergraduate Computer Science Curricula." I laid out the argument for including operating systems in the core and outlined the principles we recommended. I also argued that there would be a trend to include more systems courses in the core. Much to my surprise they selected that paper for the best paper in the conference. I guess the idea that the core of computer science contains a system course resonated with the mood of the times.

I think this answers your question about when I started getting involved in computing curricula. I got involved immediately after ACM Curriculum '68 and my first contribution was to help out with the operating system core course. I didn't stop being interested at that point. I did other things in next ten years' time. I was a member of the ACM education board, I helped get SIGCSE founded, and I was active in promoting computer science through the National Science Foundation.

Interviewer: Yes, that answers the question. Say more about promoting computer science through NSF.

Denning: In the late 1970s, there was a lot of concern in the computer science field because it seemed like we were losing systems faculty to industry way faster than we could recruit replacements. Systems faculty worked on operating systems, database systems, network systems, graphics systems, robotics systems, and a few others. Industry needed people who could design and build systems, and they could offer these faculty much better salaries. We were worried that we would not be able to teach systems courses or advise students in systems oriented projects. The process was seen as a brain drain that threatened the health of the young field of computer science.

Computer science in universities had a great history of contributing to early computing systems -- for example, the projects at MIT, Harvard, University of Pennsylvania, Princeton, Cambridge, and Manchester. Universities were very proud of their involvement in the birth of the field. In the late 1970s it seemed like the people who could be involved in the next wave of innovative systems were being recruited to industry. The computer science departments could lose their engineers, developers, and experimenters, leaving only the mathematicians to teach computer science. I have nothing against mathematicians, but with many others including mathematicians I was concerned about the balance on the faculty and our ability to grow the discipline and over the core material, which by then included a lot of systems.

Jerry Feldman at University of Rochester organized a small team to prepare a report on this problem. NSF supported their work. They examined what they called experimental computer science and raised a red flag of concern that that part of computer science would wither away from the universities. They called on NSF to help stem the drain with new programs and initiatives. They also called on university promotion committees to look for wider indicators of peer acceptance than journal publications because the experimentalists published in software magazines, distributed their software through the Internet, and showed their software at major conferences. They wanted experimentalists to have an equal shot at promotion and tenure in an evaluation system that favoured the mathematical and theoretical.

The Feldman committee showed how this collection of issues was conspiring to support the systems brain drain. They called on NSF ought to way to make experimental computer science research respectable in computer science departments, and help prevent this brain drain from getting worse.

I was elected ACM President in 1980, soon after the Feldman report. I asked the ACM Executive Committee wanted to take a position, which they did, endorsing the Feldman Report. In 1980 at the Snowbird meeting of the CS department chairs, I joined with a few other leaders and helped produce a consensus report from the meeting, “A Discipline in Crisis.” It was published in the CACM in June 1981. By that time the leadership at NSF had heard these and other loud voices and said they wanted to help. The NSF created the Co-ordinated Experimental Research Programme, CER. They sought proposals from university coalitions to undertake basic and applied experimental research in computing systems.

With NSF behind experimental computer science, there was a strong positive response from the universities and eventually the brain drain was reversed.

As part of its determination to help the CS community, the NSF took a big interest in the CSNET effort. I joined with three other department chairs to advocate that NSF and DARPA find a way to make the ARPANET available to all universities, not just the handful who had DOD contracts. We worked for 18 months to build a community consensus behind a proposal that NSF accepted in 1981. By 1986 CSNET had grown to connect all CS PhD-granting departments and research labs in the US and Canada, and a few in Europe, including altogether about 50,000 faculty, researchers, and students. CSNET connected them with a network based in ARPANET technology with different bandwidths according to the needs and means of individual departments. Small departments with limited budgets connected by PhoneNet and large departments connected by means of a TCP/IP we developed to run on the GTE Telenet X.25 packet network. CSNET provided a coordination center (the first ISP) and organized a consortium of its members to manage the network. By 1986, CSNET was completely self-supporting from dues collected from its members.

The CSNET success gave NSF the confidence that it could engage in networking projects. NSF embarked on the NSFNET project, which eventually became the backbone of the Internet and brought networking to all scientific research. NSF was a major factor in the growth of computer science and the early Internet.

A little-known historical fact is that CSNET negotiated an agreement between NSF and DARPA to allow NSF contractors to use the ARPANET. That was the first time the ARPANET allowed non-DOD traffic. It laid the groundwork for NSF to work with industry and eventually allow commercial traffic on the NSFNET backbone. By the time the World Wide Web began emerging in 1993, the principle of commercial traffic was established and enabled the quick adoption of the Web by commercial companies. The Internet (and Web) ceased to be a domain solely inhabited by researchers; it became a flourishing medium of commerce.

So in the decade following the Feldman Report, NSF and the CS departments produced quite a transition, putting computing in the middle of many innovations in research, networking, and commerce. CSNET was a key bridge from the old, closed ARPANET to the fully open Internet. CS faculty and researchers played key roles in inventing the technologies and participating in the many international committees working for their adoption. The mood of the CS community transformed from dejected and discouraged in 1979 to update and ambitious in 1989.

Interviewer: If everything was so upbeat, why did you sound an alarm that led to the 1989 report?

Denning Although we had turned the tide and were receiving new resources, hiring systems faculty, and getting involved in challenging and innovative research, the prevailing mood in the CS community bothered me. In 1984 I gave a speech at Snowbird, “Ruminations on Education”, in which I called on my colleagues to give up the sour mood and take advantage of the new environment. My ruminations were published in IEEE *Computer* magazine in May 1985. I said that the traditional programming-heavy model dominating CS curricula was reaching the end of its useful life and we needed a new model that brought forward the principles of computing in all areas, not just programming. I also said that we needed to cultivate good relations with other fields, notably engineering and sciences.

The ACM Education Board (chaired by Joe Turner) was intrigued by this and asked me to organize an ACM/IEEE committee to examine computing as a discipline and suggest new directions for curriculum.

I organized the panel and we started work in 1987, finishing in 1988 with a report “Computing as a Discipline”. We noted that it was hard to distinguish computer science and computer engineering, which was why many people at the time referred to our field as CS&E, computer science and engineering. We abbreviated this to “computing”, a term that stuck and became equivalent to the European “informatics”.

We noted that computing has deep roots in mathematics, science, and engineering and yet is different from each of those fields because of its special focus on information processes. We believed that computing had grown into its own discipline distinct from its roots. We proposed a map of the field in the form of a 9x3 matrix -- 9 rows enumerating core technologies and 3 columns for theory, abstraction, and design (the three roots). In each box of the matrix we made detailed entries of the concerns, accomplishments, and literature. With this map we were able to answer the nagging education questions of the day, is computer science engineering? Science? Mathematics? Where does it fit in a university?

This was the first time ACM and IEEE spoke with one voice about the field. ACM and IEEE have cooperated ever since on joint computing curriculum recommendations, notably in 1991, 2001, 2005, and 2013.

That is the genesis of our report. It began when spoke out against the mood of dejection and resignation. I just did not want us to become the victim of other people's stories about us. There was so much we could do for ourselves. I wanted to help computing find its own voice.

I think that our report was the beginning of finding our voice. We were able to say who we are, why we are new and not part of older more familiar fields. I think other people began to see what was different about computing and why we are not a subfield of mathematics, science, or engineering. We certainly have much to offer to mathematics, science, and engineering, but we are different because computing deals with information processes and machines that transform them. No other field has that as a focus of concern.

We also felt that our report gave grounding to our claim that computer science is *not* programming, contrary to a myth of the day. I might note that the myth that "CS = programming" has come back and in my opinion we need to fight against it.

Interviewer: Was that myth a problem for the field?

Denning I think so. Whenever someone asked "What is computer science?" our main answers were about programming computers. Many in our field celebrated great programming as the epitome of computing. Within the field, we all understood that this was an ideal, celebrating our greatest algorithm designers and system builders. Most of the progress in our field did not come from advances in programming, and yet our answers made it sound like we believed they did. We did not know how to speak in broader terms about what we do.

On top of that, the US and UK Labor Departments, which had been slow to recognize any computing occupations, began to list computing job titles. To the extent that they considered "programmer" to be a job, they defined it as a "coder" -- someone who writes the code in a language representing someone else's design, compiles it, and debugs it. The official public definitions of programmer came to mean coder, which was much, much narrower than what our ideal of programmer was.

Without realizing that the word "programmer" meant something very different to us than to our non-CS listeners, we continued to speak about CS being programming. Our listeners thought that all we did was code and occasionally advance technology. They knew we had theoreticians, but thought of them as mathematicians rather than computer scientists. They knew we had computer builders, but thought of them as electrical engineers. They did not think we were much interested in science.

I think our report gave us a way of talking about our discipline that made clear we have strong elements of mathematics, science, and engineering, blended in a new way, and that we are not simply coders or technology hackers. We wanted to overcome the disconnect between the public view of computing and the real guts of our field. Characterising the field as a field of programmers is just a giant mistake.

Interviewer: Yes. I'm curious about the composition of the task force. How did you choose the members? How did thy get involved?

Denning: The ACM Education Board chartered our committee. As I noted they were intrigued with the ideas I proposed in my "educational ruminations". They wanted us to do a report that would define a conceptual framework about a separate discipline they could use in the next curriculum revision planned for 1991. They felt that the discipline was outgrowing the 1968 framework.

Joe Turner, chair of the Education Board at the time, and I drew up a list of people we could invite to the committee. We wanted people who resonated with the idea of our field being a discipline and would propose a framework that would be useful for many years. We also wanted to reach out to IEEE and have them join the effort; we were all thinking of the field as "computer science and engineering" and we could hardly leave them out. Once we agreed on the names, Joe and I set out to invite them. I think everyone we invited joined us. The IEEE Computer Society decided that they would send a single representative since they viewed this as primarily an ACM committee. They chose Mike Mulder. Mike and I were already good friends. Our friendship helped promote good will between the two societies, which was essential later to get their endorsements for the report.

Once the committee got rolling, we sought feedback from anyone and everyone who might wish to comment on the ideas that we were examining. I remember distributing draft reports and collecting a lot of comments. I also remember a Town Hall meeting at a SIGCSE conference where we presented the draft and took extensive notes on the audience's reactions and suggestions. While small, our committee benefited from the thoughts of many others.

Interviewer: You mentioned trying to gain recognition for architecture and systems. But I was also wondering, was there a pedagogical component to the committee's work?

Denning It was intentionally minimal. We felt that our primary charter was to articulate a framework for the discipline, not to design a curriculum. The job of designing a curriculum would fall to the ACM curriculum 1991 committee after we were done. We did bow slightly to the pressure to say something pedagogical by outlining a first course in computing based on the framework we developed. We de-emphasized that part by putting it into an appendix as a speculation for a possible first course. The 1991 curriculum committee subsequently designed a more detailed recommendation for a first course based on that example. After our report was released, there was a lot of interest in that appendix, more than we wished. We wished people would instead pay attention to the framework and not worry yet about the first course.

I believe that our fear about going into course content was justified. A few years later there was a big argument about whether our idea for a first course was sound. We speculated about a first course that would survey the field and orient students to its many parts. The Curriculum 1991 committee worked with our speculation to propose what they called a “breadth first” approach to contrast with the more traditional first course in programming, which they called “depth first”. The tradition inherited from Curriculum ‘68 was that we would start our students with a lot of programming, achieving some depth at that core skill; once they got programming under their belts, they would branch out to other parts of computer science. We did not actually propose a breadth first approach; that was the idea of the Curriculum 1991 committee. The critics called the 1991 approach “a mile wide and inch deep” and claimed it would produce shallow students who could not handle the more advanced topics later in the curriculum. I think people are still arguing over that distinction to this day. Our intuition was to stay away from that, concentrate on the framework, and confine our comments about pedagogy to an appendix on a possible first course. Our speculation was our answer at the time to the question, How would we introduce the field of computer science if we only could do it in a single course?

Such was our pedagogical component.

Interviewer: It sounds like you were not entirely happy about the emergence of this breadth first terminology.

Denning: We were always interested in computing as good science and good mathematics, and as applications with good engineering. We did not invent the breadth-first idea or even come close to discussing it.

It always seemed to me that if we were going to design a curriculum based on the framework we proposed, we would have to start with an introduction to the framework. The curriculum that followed would then progress deeper and deeper into the topics. That is what we had in mind when we speculated about a first course.

If you use the same logic with the older tradition of a programming heavy computing curriculum, it makes sense to start with programming and progress deeper and deeper into it. Other topics would eventually grow out of that core for students who had acquired sufficient programming experience to handle those topics.

The curriculum 1991 committee may have tried to preserve a lot of the traditional structure in their curriculum. In that case, replacing the programming focused first course with a computing-field focused first course would indeed seem like a switch from depth-first to breadth-first. But then you would have a mismatch between the rest of the curriculum and the first course.

I think that the ensuing debate was not so much about the terms breadth and depth, but about the philosophy of the curriculum itself. Do we want to emphasize programming and get the students deeply into the practice of programming? Or do we want to emphasize the science and engineering and view programming as one of several important computing practices? I think that debate is still going on today.

Interviewer: Based on the history of computer science, I could see why your committee wanted to recognize the three historical roots in mathematics, science, and engineering. How did you get from these ideas to the terms you actually used, theory, abstraction, and design?

Denning

Adopting the term theory was pretty easy since there is obviously a lot of theory in computing and there was vocal “foundations of computing” community constantly reminding us of that. We associated the term with the paradigm of mathematics.

The term abstraction was a compromise. We wanted to include the science root. Science includes a lot of modelling. Models are abstractions. The word abstraction already had a lot of appeal within computing. So we chose that term and associated it with the paradigm of science.

The term design was actually a new insight. I remember a meeting in which Mike Mulder was shaking his head on account of all the attention we had been paying to theory and abstraction. He said that emphasis will never fly with his engineering colleagues. Then he said, “Design would work. It is a deep value of engineering and appears in the accreditation criteria. It is integral to software engineering.” That moment of insight caused the entire committee to coalesce on design rather than engineering, and we sketched out a design paradigm.

We used the term paradigm more like a “process of thought” rather than a belief system as it was more commonly understood. We sketched each paradigm with four simple steps and displayed them side by side. We said that computing combines all three paradigms. We said that the combination is unique among disciplines and is a distinguishing feature of computing. We also noticed that each paradigm included steps where the other paradigms might guide the step, for example, when an engineer runs a model to guide a design decision.

We were all very pleased with this formulation. Everyone in computing seemed to identify with at least one of the three paradigms, and could therefore see that a substantial part of the field aligned with what they thought was important. Mike Mulder reported later that his IEEE colleagues accepted this formulation and from that moment we had a solid basis of working together.

And after that, we were all comfortable with the term computing for the whole of the framework, rather than “computer science and engineering”.

Interviewer: It makes sense.

Denning: It is interesting that some of the key ideas in the final report emerged as “accidental insights” in our conversations. One was the design insight we just discussed. Another is the 9x3 matrix.

We were in the process of drafting a presentation for a Town Hall meeting and were having difficulty figuring out how to get all the ideas we have come up with into a few clear slides. We had to draw the slides by hand -- no Powerpoint at the time -- and so being clear and concise was of great value.

Allen Tucker came up with the idea of displaying all the pieces in the form of a 9x3 matrix. He said something almost offhand, “How about we display it as a matrix?” The rest of us instantly saw that this provided a nice image that connected the parts. Our Town Hall presentation fell quickly and neatly into place after that.

Interviewer: I see. (Laughter). You were just talking about working with Mike Mulder from the IEEE side. I know that the ’91 report was the first curriculum report on which ACM and IEEE worked together. Was that collaboration an outcome of your committee's work?

Denning: Yes, indeed. The collaboration we started then continues to this day. All the subsequence curriculum recommendations and updates – 1991, 2001, 2005, and 2013 -- were collaborative efforts between ACM and IEEE. Mike Mulder and I were both "collaboratists". We both believed that we do better work when we think together rather than alone. I have been immensely gratified to see that the two societies wanted to continue the collaboration begun at that time. I, myself, have always done better work when I operate in that mode. Even my sole-authored papers benefitted from extensive consultations with other people.

In the early days of the collaboration, Mike was the main point of contact for IEEE and I for ACM. We both turned our networks in the two societies for readings and reactions on the latest ideas we were considering in the committee. Over the years we broadened way beyond two points of contact interacting. Now the entire boards do that.

When we finished, ACM published the report as a booklet. We made an executive summary and published it in CACM in January 1989. The IEEE Computer Society published a condensed version in *Computer* magazine in February 1989.

Interviewer: Okay. That takes us up to late 1980s and early 1990s. Let's talk about your perspectives on the more recent efforts, I believe you were the Chair of the Education Board during the 2001 report and you had retired from that post by the time of the 2005 update. Then of course was the major update in 2013. What was your involvement in those reports?

Denning: From my perspective the 2001 report was a more or less routine update to a curriculum recommendation. It was chaired by Eric Roberts and Russ Shackelford. It was also a collaborative effort with the IEEE. They used a lot of Town Hall meetings to give progress reports, gather people's reactions, and find areas of consensus.

They expanded the list of core areas of computing from the 9 we had in 1989 to 14. They drew up a summary of the 14 areas and their major sub areas, listing 130 sub areas in all. Since it would be impossible to cover all the sub areas in a single four-year curriculum, they took votes to find the consensus on what areas absolutely must be included in every curriculum. That shorter, must-have list was about 60 topics -- still a lot. What emerged was a much more detailed picture of how a computer science department could embrace a computer science curriculum.

The 2001 committee also recognised that there were several related disciplines under the single computing umbrella -- computer science, computer engineering, software engineering, information systems, and business oriented computing. The ACM and IEEE followed up with an update in 1995, in which they offered give separate volumes customized for each of these five sub disciplines. They achieved a pretty comprehensive set of recommendations for everybody in computing.

The more recent report, in 2013 one, in my view, followed a similar pattern of identifying major areas and sub areas. The field continued to expand. They showed 18 major areas and 175 sub areas. Through a consensus voting processes, they classified the sub areas into must-have, nice-to-have, and optional. They also recognized that computing interacts with many other fields and gave recommendations on how to organize those interactions.

You can see the growth at a finer level too. Take a look at the operating system course. Today one of the most popular books is the Stallings book, which over seven editions has grown to about 700 pages. In the US the book costs $160 for the book. There is so much material, the students simply cannot grasp it all. The same thing is true with the algorithms book for the algorithms course; it's in its fifth or sixth edition now, and it just keeps getting thicker and thicker, all great stuff, but students cannot grasp it all. The standard texts for single courses have reached this thickness of Allen Tucker's *Handbook of Computer Science*, or Anthony Ralston's *Encyclopaedia of Computer Science*. Individual textbooks have become encyclopaedia in their own slices of computer science. We've go to do something to help our students grasp all this.

Interviewer: Wow. What is the significance of this growth?

Denning: These curricula clearly demonstrate the growth of the computing field over the years. They confirm our sense that the field is too large to be completely covered in a four-year curriculum.

It has been interesting to me that the names of the major areas in these curricula are all core technologies. These are the technologies we have grown up with and perfected. Our accumulating picture is a field of technologies in rapid growth.

I think this accumulating picture has been an impediment to our acceptance by other fields as a legitimate field in its own right. If the others cannot see past the technologies to the base principles of the field, they will persist with a perception ours is a technology field but not a science. They say, “These guys are just technologists. They have a few good mathematicians and engineers, and maybe even rarely a scientist, but they’re basically technologists.” I think this is the source of the continuing question, “Is computer science a science?” I’ve heard critics in other fields saying that computer scientists are really technologists misusing the term science. They think of computer scientists as advanced programmers, but not as scientists.

For this reason I believe the historical progression of focus on computing as a series of technologies has begun to outlive its usefulness. It’s certainly true that computing has been a driving force in technology advancement and the agent of many major advances and innovations. We do not want to throw away the technology history we are. But my fear is that our curriculum has gotten so technology oriented that it’s short-changing important parts of the field, especially the many growing interactions with other fields and the rising importance of design in our field.

I also see the architecture and systems part getting overwhelmed by the programming part. You may have noticed in the last three or four years there has been an explosion of interest in “coding academies”. It is pretty amazing -- kids all over the world are signing up for hackathons, hour of code, coding clubs after school, coding weekends, coding summer work camps, and more. It is great that, after so many years of our wrestling with student non-interest, computing is catching on in a big way.

The rhetoric that comes this growth is expansive. It says that coding is an essential part of the future of civilisation. Coding is the language we will need to live in a world dominated by computing technology. We need to start learning coding in middle school age, if not sooner. If coding is not already offered in those schools, it should become a formal part of the school system.

This coding movement appeals to our collective desire to know more about the way digital technology is shaping the world. We want to be part of it and to shape it.

Interviewer: Is there a downside?

Denning: I worry that this is a resurrection of that old story, “CS equals programming”, which is now being retold. We fought hard to dispel this myth in our 1989 report because it was casting our field as much narrower than it really is and it was hampering our efforts to establish peer relations with other fields seeking collaborations with computing. I am concerned to see it coming back.

The current version of the story is now focused on coding, not even the ideal of programming we discuss within computer science. This story claims that coding is the basis of computer science, and that learning coding is all you need to get access to the rest of computer science. Coding opens the door to computer science for kids. To me, there is a steep staircase just after the door opens to computing. Opening the door does not help you up the stairs. There is so much more to computing than coding. What comes after the coding academy? How do we help our young people fulfil their interest in computing?

Along with this positive change of mood toward computing among young people has been a genre of new popular books focusing on the way that computer technology has advanced and shaped the world. These books show a special reverence for the algorithm. For example, John McCormack of Princeton wrote a book “Nine Algorithms that Changed the Future”. It’s a really nice summary of nine algorithms that most people have heard of in one way or another. However, behind the book lurks an assumption that programmers and algorithm designers are the driving forces behind progress in computing technology that all the great computing advances have been algorithm breakthroughs. An example is the claim that Google’s page rank algorithm changed the world.

If you take a closer look at what Google actually did, you see that they spent a long time and a lot of investment building their data warehouses, which are the bases for the fast searches we now see and are the platforms for other Google services. Google developed the MapReduce paradigm and the distributed computing platform to support it. They would not be the success they are today without that platform. In so doing, they pioneered new directions in computer architecture and operating systems. Their architectures are important for the analysis of “Big Data” and for “Cloud Storage”. None of those advances has anything to do with the page rank algorithm. Operating systems, networks, architecture, and data management are all part of computer science, but they are not advances in algorithms. The story “Page rank algorithm changed the world” is quite misleading. It is the beginning of the Google story but is a small component of what Google has achieved.

I’m seeing hints that students are getting disillusioned by the gaps between the expectations these stories generate and the realities of the computing field. The algorithms stories are like hero stories where the rest of the world disappears and you do not see the support system that makes the hero possible. One of my co-workers has a son who went to Berkeley’s CS department. He was turned on by the summaries of the magic and joys of computing presented to him in his first course by Dan Garcia. Then her son found himself buried for two years in courses that demanded intensive programming. He was transformed into a good programmer but that was not his real interest and he began to resent it. He wanted to study artificial intelligence and human interaction but was disappointed that he did not have the time or prerequisites to take those courses until he was well into upper division. Finally, he switched majors to cognitive science and is loving it.

I know this is an anecdote, but it worries me that the heavy emphasis on programming in computing curricula can drive good students away. Programming is an important part of computing, but my colleague's son wanted to defer some of the programming so that he could study the things that turned him on, and he could not.

You have probably heard senior leaders in the field calling for more attention to design. This is motivated by the need for reliable and dependable large systems, and for systems that support the everyday practices of their user communities. How to build such systems from thousands of components and with large numbers of coders and implementers involved, is perhaps the most challenging area of computer science. The problems of design are much more challenging than those of computational complexity. Very few computer science departments deal with design; very little is said about design in the ACM curriculum recommendations. An overemphasis on coding and programming distracts from issues of design, and I think that's going to come back and bite us.

You have probably also heard about the severe problems of the CS Advanced Placement Curriculum managed by the Educational Testing Service. The idea is that getting a good score in the AP exam will get you credit for the first computing course when you go to college. Since the first computing course is about programming, it would make sense that the AP curriculum and test are about programming. Around 2000 there was a committee formulating just such a new AP curriculum based on object oriented programming. I was chair of the ACM Education Board at the time and we sent several representatives to the committee. We endorsed their recommendations to build AP around object oriented programming. Well, it took a few years to roll that out, and it was soon apparent that it was a disaster. The Education Board had not made a good call. Teachers who hardly knew programming were being asked to teach advanced programming; they just could not do it. The CS AP curriculum and exam have become immensely unpopular among teachers and their students. There is a new committee now building a new AP recommendation around "CS principles". It will be a few more years before that is rolled out. I hope that the principles are more than just programming principles. I'd like to see architecture and design principles covered too.

Interviewer: Oops.

Denning: Yes, that object oriented AP was a sad story. The entire Ed Board was seduced by the argument and supported it. The problem was that we overestimated our ability to communicate a rather sophisticated and complex technology in a way that students and their teachers could grasp. Teachers found they could not understand the material and its nuances, and there were few training workshops to help them come up to speed. From the perspective of the teachers, object oriented programming was a much more advanced concept than we thought.

So the teachers and their students did not get it. This AP curriculum became very unpopular very rapidly. Jan Cuny at NSF was concerned that the AP was not attracting students and teachers. NSF had a goal of qualifying 10,000 new computing teachers for the K12 schools; that AP curriculum was scaring them off. NSF sponsored the development of five pilot first courses based on CS principles and helped persuade the Educational Testing Service to commission a committee to design a new AP curriculum around CS principles. Dan Garcia's course at Berkeley was one of the five pilots. I was very glad to see this happen because I believe that a curriculum based on computing principles will be more effective than one limited to technologies.

I do however, have a concern over the choice of principles. These pilots emphasized programming and coding principles, such as recursion and divide-and-conquer. They had very little about systems and nothing about design.

As you know from the previous conversation, I've long been a champion for computing principles and I've always sought framework of principles that covered everything from algorithms to architectures and design. I believe that a framework that emphasizes programming is imbalanced and plays into that unwanted perception that "CS = programming". I have been promoting the Great Principles project since 2004 in order to demonstrate a balanced framework. My book with Craig Martell. *Great Principles of Computing*, was just published by MIT Press. The computing field is big and it keeps getting bigger; it has a very deep set of principles and they're not all programming principles. You really can't understand the field without understanding the whole set of principles. I hope we are not heading for a train wreck when a narrow external perception of our field crashes into the real complexities of computing and design.

Interviewer: Why would you speculate that such a train wreck is possible?

Denning: It just seems to be a drift. I think helps to understand the drift if you step back and view our history. We have grown up in a machine age, where our technologies, not just computing, have become better and better at automating tasks. In computing, we have seen the progression of bigger and bigger and faster and faster computing machines. Today's epitome of that progression is giant supercomputers. China, US, and Japan have been trading positions as the country with the fastest supercomputer. These machines carry out about 10 to the 15th power operations per second, or 1 peta-op. Ten years ago we thought such speeds were just dreams. These machines are used for the biggest and hardest computational problems we have, such as weather forecasting, climate modelling, oil exploration, aircraft design and simulation, and hunting through huge data sets for faint signals. These machines perform deterministic computations and have no intelligence.

These machines are among the billion or so machines of all kinds connected together in the Internet. The machines and network connect over 4 billion people, and the numbers continue to grow. The Internet is a constant dance between humans and machines, amplifying each other. In this network new kinds of computations are emerging, such as crowd sourcing, ride sharing, temporary rentals, and the so-called sharing economy. These networks now perform tasks that a decade ago we had no idea were possible.

I call this growing network of machines and humans “the organism” because it behaves more like a living organism rather than a machine. The organism is becoming part of our lives with more and more people connecting with each other using their mobile devices. The organism is not deterministic. It is intelligent -- the combined, amplified intelligence of four billion people. It is unpredictable and the world has much more uncertainty than ever before.

We are trying to learn to navigate in the organism using an understanding of computing that belongs to the machine age. One of my favourite examples is our belief that we are now capable of building accurate, large scale models of global systems and solving them precisely with our advanced machines. In other words, we believe that the machines give us the capability of overcoming uncertainty. But exactly the opposite is true. The organism is more uncertain and less predictable and no mathematical model will overcome that. I think that a lot of people hope that they can solve the increasingly hard problems in the world with better mathematical models and more powerful computers. They put a lot of faith in their ability to design models and algorithms. That belief is impeding us from learning how to navigate in the organism with all its inherent uncertainty.

The world we grew up in is all about machines. It is natural that the computing curriculum is technology oriented and focuses on controlling the machines with good programs. We have not yet developed the skills to function in the organism and we are hoping that our standard curricula will prepare our students for that. I’d like to see a new curriculum effort that recommends a computing curriculum for the organism age.

When we see drifts in the world we want to apply computing technology to channel the drifts in better directions. But these drifts are made of many unpredictable behaviours of individuals and machines. We need to be looking not at control or precise prediction, but at channelling and shaping the drift. We need to emphasize design that responds to concerns rather than programming that builds control systems or automates routine tasks. Few of our curricula pay any attention to design.

I know it's hard to talk about this. My mind is also a product of the machine age and I find it hard to think inside the new world of the organism. Many of our young people are moving more naturally with it than I, probably because they do not have my history with the machine age. I see young people gravitating toward design and getting disgruntled with programming-heavy curricula.

So that's where I think the future of education needs to be looking, aiming to understand the organism age and help our students escape from what seems to them to be the prison of the machine age. I am eager to help out with these new investigations.

Interviewer: This is very helpful I do appreciate your unique perspective, from the very beginning up until now, going beyond. Thank you.