

Re-centering Computer Science

Position Paper for ICER06

DRAFT v2 -- 1/4/06

Peter Denning
Naval Postgraduate School
pjd@nps.edu

Andrew McGettrick
University of Strathclyde
andrew@cis.strath.ac.uk

INTRODUCTION

The enrollment drops in computer science and related degree programs concern us all. Many of us suspect this decline is not seasonal, but signals deeper reasons that make our field appear unappealing to prospective students. The deeper reasons cluster around an external image of our field that “CS=programming”. We’ll examine curriculum structure and teaching practice as possible generators of this image and suggest alternatives.

EXTERNAL IMAGE

A persistent theme in the analyses of why young people choose fields other than computing is a perception that computing is mostly about programming. Most computing people understand “programming” to mean the discipline concerned with design, development, testing, debugging, documentation, and maintenance of software. Along with analysis of algorithms and complexity theory, many of us take programming as the heart and soul of computing. In recent years, the Bureau of Labor Statistics has co-opted the title “programmer” to mean something like “coder” and not this full range. The result has been a narrowing of the external understanding of the field. In this light, what we internally thought of as *broadening* to object-oriented programming was perceived externally as a *narrowing* to the Java language.

We have known for a long time that our external image is “CS = programming.” In light of our self-understanding of what that meant, most of us were not concerned by this. But now that we see outsiders interpreting this image narrowly, we are concerned, and we are looking for ways to change it. This myth is now hurting our field and our efforts to attract students. What can we do to reverse, once and for all, the CS=programming myth? What would be involved in changing this view of our field?

Some of the answers to these questions are outside the scope of our workshop -- for example, better public relations, involvement of computing people in high-profile, glamorous projects, inspirational career profiles, and working with CSTA to achieve better understanding of CS in high schools. One of the answers -- our curriculum, organization and operation -- is very much within our scope. Our curriculum is our public statement of how we perceive our field and ourselves.

Over time, we can influence our external perception by altering the structure and organizing themes of curriculum.

The dominant organizing theme in our curriculum today is programming (in the large sense noted above). Programming, however, is not the only coherent organizing theme for a computing curriculum. Seven others are:

- *Great Principles*: Organize around the fundamental principles of computing. Our initial work in this area finds six categories of principles: computation, communication, coordination (interaction), recollection (storage), automation, and design. These six timeless categories are easier to approach than the current taxonomies of 30 or so constantly-changing core technologies. An ACM Education Board task force is seeding a great-principles library. (Denning 03)
- *Innovation*: Organize around innovation. Study the great innovations of computing, their fundamental principles, and their linkages to other fields. Teach the foundational practices of innovation. Show how this brings out the best of computer science. The recent ACM Job Migration Task Force report strongly endorses this organizing theme. (Denning 04, Denning-Dunham, Denning-McGettrick)
- *Cross-Discipline*: Organize around relationships between computer science and other fields. Many of these fields are currently producing rapid technological advance enabled by computing. The most interesting and successful advances of computing are products of such past interactions. (Rosenbloom)
- *Experimental Science*: Organize around experimental computer science. This brings students in contact with hands-on experiments on systems and with the many fascinating and challenging systems issues of computing; and with the realities of emerging technologies. (Snyder)
- *Game Development*: Organize around multiplayer learning games used for training, testing and evaluating organizational processes, and advanced entertainment systems. (Zyda)
- *Knowledge Management*: Organize around information management, quality checking of information, workflow, databases, and some intelligent agents. Address crucially

important social aspects such as discovering the value, location, retrieveability, and usability of information.

- *Human Computer Interface*: Organize around the theme of computers and humans interacting in synergistic systems to accomplish tasks neither could do alone.

We think it would be valuable for the ACM Education Board, working with NSF and others, to discover which of these has a following and develop suggestions on how individual departments could take up each theme. Let the departments choose for themselves and share their results with the rest of the community.

TEACHING PRACTICE

A recent article by Marcella Bombadieri in *Boston Globe* (18 Dec 2005) examined the growing gender gap in computer science. In the middle 1980s, 37% of CS undergraduates were women; today it's 6%. What is interesting is her speculation about why this might be so.

In the early 1980s, interest in computer science started to explode. The Internet and personal computer were just starting to be widespread and many people saw computing as a field rich in opportunities. Computer science departments were flooded with students but were not given additional resources for the additional load. In fact, the problem was exacerbated by a "brain drain" of systems people exiting universities for industry. The response of many departments was to clamp down by making courses technically more difficult and by insisting on strong math skills. The increased fascination with pure technology and abstraction distanced computer science from applications. The lack of connections to other fields and to useful applications significantly reduced the field's attraction to women. Not until the dot-com bust in 2002 did it begin to be evident that this highly technical and insular teaching practice was a liability.

CONCLUSIONS

We do not believe the enrollment drops are a simple seasonal phenomenon that will take care of itself if we are patient. We believe that the enrollment drop calls us on to examine both our self-image and our public image. As challenging as a hard look at our self- and public images may be, the situation is a real opportunity for us. We have the opportunity to think through how we want to be perceived by other fields of science and engineering, by the general public, and by prospective students. And then to organize our actions to produce the image we want to project.

Our thesis is that the myth CS=programming comes from our internal view that programming is the core of computer

science. Internally, we assign programming a rich and broad meaning. The theme of programming permeates the design of most of our curricula and our ACM curriculum recommendations. Externally, however, programming is seen as narrow and susceptible to migration off shore, and thus not a promising future for a computer scientist.

We propose the possibility that the curriculum structure, which reflects a programming-oriented image, is a main reason for our narrow public image. Ingrained teaching practice can also produce the narrow external image. In fact, structure and practice are connected. Current teaching practice concentrates heavily on programming theory and practice. It is very technical and mathematical, and its connections to other fields and applications are cursory at best. A department that chooses one of the alternatives suggested above would necessarily alter its teaching practice. A department that decides it must alter its teaching practice would find these alternatives to be fruitful sources of new practices.

The ACM Education Board can help by developing advice for those who want to organize their department's curriculum around any of these alternatives. The NSF can help by supporting departments who will experiment with them and measure their results.

We intend our proposals to heighten the motivation of students, to give computing a more human and less abstract orientation, and to encourage intellectual diversity.

We're not trying to eliminate programming, just to dispel the impression that we are only programming.

BIBLIOGRAPHY

- Denning, P. Great Principles of Computing. *ACM Communications* 46, 11 (Nov 2003), 15-20. See also <http://cne.gmu.edu/pjd/GP>.
- Denning, P. The social life of innovation. *ACM Communications* 47, 4 (April 2004), 15-19.
- Denning, P., and B. Dunham. Innovation as language action. *ACM Communications* 49, 5 (May 2006), to appear. See URL <http://cne.gmu.edu/pjd/PUBS/CACMcols/cacmMay06.pdf>.
- Denning, P., and A. McGettrick. Recentering computer science. *ACM Communications* 48, 11 (Nov 2005), 15-19.
- Rosenbloom, P. A new framework for computer science and engineering. *IEEE Computer* (Nov 2004), 31-36.
- Snyder, L. et al. *Academic careers for experimental computer scientists and engineers*. National Academy Press (1994).
- Zyda, Michael. MS in Computer Science (Game Development). <http://gamepipe.usc.edu/Masters.html>