The Computing Profession at a Crossroads

Venkat N. Gudivada, University of Michigan, Dearborn

By any measure, the computing field enjoyed a golden era during the past decade. About two years ago, however, a rather dramatic meltdown began as harsh economic realities unfolded, causing an attendant upheaval in the field. Suddenly, there was a glut of seasoned computing professionals in the job market, and businesses became disenchanted with the value of their computing investments.

This newfound wisdom prompted several questions and concerns about return on investments, software's appalling quality, the negative impact of immature and fleeting technologies, and the deeply entrenched practice of software development as art, not science.

ACADEMIC IRRELEVANCE

Empirical evidence reveals that fewer than 20 percent of computing professionals in the US's financial services industry have any formal academic background or training in computing systems. Over the past decade, project managers ignored fresh computing-discipline graduates, instead hiring crash-course training-center or liberal-arts college graduates with a couple of years experience in the industry. These managers held the view that recent computing graduates needed to go through long company-sponsored training programs and apprenticeships before they could perform useful work.

Industry and academia must work together to chart the best course for the profession's future.

On the academic and training side, institutions from universities to community colleges to street corner mom-and-pop training centers expanded existing computing programs or started new ones. These programs came in many forms, partly to address the allegedly increased demand for computing professionals. Others saw this trend as an opportunity to realize their cherished dream of becoming a program director, chair, or dean.

Academia, however, failed in its educational mission in the name of fundamentals versus fleeting practical knowledge. Neither academic leaders nor professional organizations such as the IEEE Computer Society and the ACM expressed concern about the enormous impact and attendant problems caused by mushrooming computing academic programs.

The fallout from this trend included a curriculum irrelevant to overall industry needs and a lack of pragmatic and enforceable curriculum standards, quality control, resources, and an identity for the discipline itself.

COMPUTING RENAISSANCE

Today's computing profession offers four broad employment categories:

- design and manufacture of computer hardware;
- development of system software, embedded systems, and general-purpose software such as database servers, toolkits, and application frameworks;
- applications development with minimal programming, using configurable and interoperable third-party commercial off-the-shelf components; and
- application implementation using packaged software such as enterprise resource planning systems.

Only employers for the first two categories seem to attach significance to formal academic degrees in the computing discipline. The third category employs more people than the others. The fourth category demands both computing technical skills and a deeper understanding of each application domain's business processes.

In addition to programming expertise, employers expect future computing-discipline graduates to master many other skills, including systems engineering and end-to-end system architecture and its elements: security, performance, scalability, availability, reliability, supportability, manageability, and maintainability.

Programming services will emerge as

Continued on page 90

Continued from page 92

a commodity item, with most programming outsourced to the lowest bidder for economic reasons. The market's current development toolkits and frameworks already sport high-level abstractions that even some smart high school students can use to develop simple yet useful applications.

Thus, an industry renaissance in favor of developing functionally extensible applications has led developers to design applications based on an architectural framework with built-in extensibility points. These designs allow the implementation of new functionality without writing any code. Functionally extensible systems lie somewhere between approaches like parameterized and aspect-oriented programming and enterprise resource planning systems. This trend will further lessen the need for people with no skills beyond programming.

Revitalization

In addition to the recent demise of many dot-com companies, further consolidation in the computing industry will result in the disappearance of some businesses and the reemergence of those with clear vision, distinctive competence, and an effective strategy.

This revitalized computing industry will demand high quality from employable graduates who can understand and solve real problems and orchestrate end-to-end solutions. The responsibility for supplying this expertise rests primarily with academia, while industry provides an active advisory and support role.

Identity challenges

The academic front faces challenges to the identity of the computing discipline in both general and computer science. The ascent to prominence of interdisciplinary research in bioinformatics, computational biology and physics, library science, and digital libraries has only made defining, establishing, and asserting the true identity of the discipline more imperative than ever.

Some circles have even asserted that computer science is inherently an interdisciplinary subject with little core knowledge to justify its distinct identity. Yet others compare computer science with mathematics in that, just as we need applications to extract practical usefulness from mathematics theory, computer science provides the underpinning for academic disciplines in engineering and the sciences. Therefore, these disciplines can better fill the role of educating computing students and designing and developing more relevant curricula that better reflect these students' domain-specific needs. The emergence of computer science courses transformed to suit an application-oriented context in disciplines such as engineering management, systems engineering, biology, and physics vindicates this argument.

The academic front faces challenges to the identity of the computing discipline in both general and computer science.

COMPUTING VERSUS ENGINEERING

In a previous column ("Jobs, Trades, Skills, and the Profession," *Computer*, Sept. 2002, pp. 104, 102-103), Neville Holmes provided an illuminating look at how the computing profession differs from the traditional engineering branches. According to Holmes, engineering clearly demarcates the roles played by its practitioners, tradespeople, and the resulting products' end users.

The computing field lacks any such demarcation, however. Industry views the computing profession as a collection of variegated and vacillating skills with no clear structure. Holmes argues that we can correct some of the issues the discipline faces if we accept computing's status as a secondary profession, one that aids and abets other professions.

Meanwhile, software engineering is emerging as a distinct discipline whose roots lie in computer science. David Lorge Parnas offers compelling reasons to support that perspective in "Software Engineering Programs Are Not Computer Science Programs" (*IEEE Software*, Nov./Dec. 1999, pp. 19-30). Parnas cogently argues for and substantiates the merits of developing software engineering programs in engineering departments, especially in electrical and computer engineering.

THE UNCERTAIN FUTURE

Computing professionals may soon be held legally liable for a defective software system. Accountability and responsibility should thus be at the heart of any professional practice. Therefore, we face a compelling need to address issues related to curriculum structure and its standardization, its currency and relevance to overall industry needs, its program quality, and the extent to which it fosters collaboration between industry and academia.

Wake-up call

Academia must awaken to the hard realities the computing professional faces in the field and then become more responsive and accountable.

Now that the body of knowledge has matured and stabilized, we need to define curricula with much more rigor, then standardize it across all institutions irrespective of their ability to acquire the needed resources to implement that curriculum. This approach contrasts sharply with today's suggested computer science curricula models and accreditation standards. We must strike a balance between focusing on fundamentals and imparting practical skills. Unfortunately, many departments lack the resources for accomplishing this task.

Further, in computer science instruction, a dichotomy exists between theory and practice. Professors frequently take a concept-oriented approach to teaching, compartmentalizing any practice-based learning. Often, capstone courses in computer science seek

to bring coherence and consummation to the knowledge students have gained in various courses—yet theory and practice should be interwoven across the curriculum.

This fundamental difference in instruction style raises the question of computer science and software engineering programs coexisting within the same academic department. A coexistence scenario is desirable because it lets software engineering programs leverage computer science programs' core competency. But several factors argue for separating the two disciplines: curriculum structure and standardization, accreditation and licensing, and culture clash. However, resource-related issues will force many software engineering programs to debut in the coexistence scenario; they will thus assert their own identity only gradually.

Meaningful collaboration

Academia and industry should collaborate more meaningfully and productively. Although this statement has been made by many, time and again, collaboration has yet to go beyond obtaining grant money and establishing inactive industry advisory committees.

Industry views the professor as an idealist detached from reality. But simply blaming academia for the field's current state doesn't ameliorate the situation. Working through professional societies to shape the curriculum, providing real-world case studies for classroom use, establishing programs through which faculty can spend summer and sabbatical time in industry, and exploring other avenues to strengthen cooperation would benefit industry.

EXPERIENCE MATTERS

Generally, academia doesn't value industry experience. Worse, industry simply ignores academic experience. This hostile attitude doesn't help the profession's advancement. It prevents attracting seasoned industry professionals with advanced degrees, especially those who have gained their practical insight by working in the trenches. Similarly, few opportunities exist for faculty to spend their sabbatical time working with industry to solve real problems.

At least part of the faculty in software engineering programs should possess substantive practical experience and insight—traits somewhat more valued in engineering disciplines than in computer science. Institutions commonly expect the engineering faculty to possess substantial industry experience because designing and developing a real-world system pose much greater challenges than writing a research paper. In contrast, computer science departments typically favor publications over practical experience.

Professional societies should champion both the need for and administration of certification and licensing.

Academia alone cannot extricate itself from the current situation without the active leadership of industry and professional organizations such as the IEEE Computer Society and the ACM. For better or worse, academicians dominate professional societies, thus these societies primarily reflect academics' needs: organizing conferences and publishing magazines, journals, and conference proceedings. Introspection is paramount.

Mission, vision, goals, and objectives must all be closely examined and aligned with reality, while balancing tactical needs with long-term strategy. Some energy should be channeled away from introducing more conferences and publications and into reducing the explosion of more or less the same information packaged differently and appearing in a plethora of forums.

CERTIFICATION AND LICENSING

Professional societies should champion both the need for and administration of certification and licensing. They should also liaise with government, business, and industry to educate these sectors about the significance of hiring licensed professionals and the process involved in that licensing.

Currently, the market treats the IT workforce as disposable labor. Societies should actively seek to improve the stature and public image of computing professionals and address issues that affect the careers of computer science and software engineering graduates and practicing professionals.

The recently initiated IEEE Computer Society's Certified Software Development Professional credential is a good step in this direction (Leonard L. Tripp, "Benefits of Certification," *Computer*, June 2002, pp. 31-33). Counterarguments to certification have been advanced, however, on the grounds that poor software quality stems from more fundamental problems than software engineers not all learning the same material (Adam Kolawa, "Certification Will Do More Harm than Good," *Computer*, June 2002, pp. 34-35).

Educators must recognize the profession's current reality (Spencer Johnson, *Who Moved My Cheese?* Putnam, 1998), forge a vision for improving it, devise a strategy for realizing this vision, then provide sound leadership for its implementation. If we do not, "survival of the fittest" alone will determine the profession's Darwinian future. ■

Venkat N. Gudivada *is a visiting associate professor of computer and information science at the University of Michigan, Dearborn. Contact him at gudivada@umich.edu.*

Editor: Neville Holmes, School of Computing, University of Tasmania, Locked Bag 1-359, Launceston 7250; neville.holmes@utas.edu.au