The Unspoken Revolution in Software Engineering

Bertrand Meyer, ETH Zurich

Software professionals must adapt to the new reality of offshore outsourcing.

The 2006 International Conference on Software Engineering's call for papers lists more than 70 topics that excite everyone in the field, including me. Just as conspicuous—and ironic given that this ICSE will be held in, of all places, Shanghai—is the absence of any reference to what may be the most important event affecting software today: the massive transfer of development activities from the US and Europe to developing countries.

There's little analysis of offshore development in the technical computer science or software engineering literature, but this phenomenon is bound to have a profound effect on the field: not only the obvious human effect, but also a long-lasting influence on the technology itself.

A look at the numbers provides clear evidence of the offshoring phenomenon's size. According to Global Outsourcing Report (www.horasis.org/press_releases_all_4.php), three quarters of US companies outsourced at least some of their IT business. IT outsourcing grew to a $100 billion industry in 2005, representing close to a third of IT budgets.

Although statistics differ on how much of this business goes to offshore locations, it appears from various estimates (www.rttsweb.com/services/outsourcing/stats.cfm) that it's a fourth to a third of that amount and will reach $150 billion in five to 10 years.

India has, in just a few years, built an entire industry of offshore software development: $17 billion last year, up from $4 billion five years ago. To cite two of the main players there, this past September Tata Consultancy Services had 53,000 IT workers and Infosys had 46,000, up 12 percent and 15 percent, respectively, from July. Several other countries—Russia, China, and many Eastern European nations—are following quickly, with many more hoping to join the club. At professional events, booths promoting Vietnam, the Philippines, and even Ghana have become common.

DECLINE OF THE WEST?

The effect on the software engineering professions in the West is visible everywhere. In Silicon Valley and other software hubs, many jobs have disappeared. One category has almost vanished: basic programming on a large scale.

For programmers without any special skills, this is the end of an era. Confronted with a project requiring many developers for fairly routine applications, any large company today, and many small ones, will outsource to a cheaper country.

There are still a few protected pockets: If the job includes an extensive human interface component, you won't find too many developers knowing Swiss German in India, and this language issue explains in part why the English-speaking world is most visibly affected. But such pockets are increasingly rare.

No wonder anxiety runs high. The statistics that tend to attract the most attention track job losses, often only projected, like the Forrester study (www.informationweek.com/story/showArticle.jhtml?articleID=20900333) that asserts offshoring by US companies will cause a loss of 3.4 million service jobs by 2015. Such estimates must be viewed with some suspicion, not just because 2015 is a long way off, but also because the results do not include any effects in the reverse direction. In any industrial revolution, some jobs disappear while some are created. What's undeniable is that offshoring has become a key software development technique and that its share will continue to increase.

Is offshoring something to be feared, or is there something inherently good in it? Obviously, for the folks in Bangalore and Bucharest it's a great development, and, at least abstractly, we should be glad they can join the fun. For some compa-

Continued on page 121

Continued from page 124

nies, it's clearly productive too. But there are also resounding catastrophes. Various studies suggest that something like 40 percent of offshore projects fail to deliver the expected benefits.

REVELATIONS

I find outsourcing so fascinating partly because it serves as a magnifier and revelator of just about everything in software engineering.

For years, software methodologists like me have told the world about software engineering principles, sometimes with effect, sometimes with—let's be euphemistic—less effect. Now comes a project that splits the team across three continents, and suddenly software engineering advice ceases to sound like preaching, becoming instead a matter of success or abject failure.

Traditionally, we've told people to do requirements engineering, and they've often skimped on it. Now, with the implementation team half a world away, the team won't produce anything decent without a strict process and requirements standards.

We've exhorted programmers to care about documentation, configuration management, quality assurance, contracts, project management practices. These are no longer just good ideas, but techniques that can each, if not done right, become project killers.

When the design is done on the 12th floor and the implementation on the 11th, you may think it's all right to cut corners on good software engineering practice and renounce choosing the tools you know are really best. When the rest of the team is half a world away, forget it.

It's not just the time differences—it doesn't help when one team is asleep while the other works—but also the cultural differences: East may get a dose of West and West may fancy a bit of the Orient, but not always the twain shall meet. More profoundly, it's software engineering: The annals of outsourcing already include heaps of project failures caused by nothing more than neglect of our discipline's principles and tools.

OPPORTUNITY LOST

Reflecting further on the fear of good programming jobs disappearing forever, I can't help thinking that the software world has missed a great opportunity. It's no secret that, for the most part, it sees itself as a service industry. I have heard this view proclaimed proudly and repeatedly. For my part, I have always thought that the phrase "software engineering" should be taken for what it implies: a systematic discipline based on quality tools.

Having for many years been in the business of selling better software technology through development tools and languages, I have noted again and again how much easier selling manpower seems to be. During the boom years, hiring one more programmer was more attractive than investing in a tool. Now such decisions are come back to haunt us: If programming is a service, there will always be someone ready to sell programmers' services at a cheaper price.

Successful outsourcing companies like to say that customers come for the price and stay for the quality.

The other approach would have been to leverage tools to increase productivity and let companies rely on a smaller number of highly qualified programmers, the ones hardly ever in danger of losing their jobs.

Quality is indeed the central issue. Successful outsourcing companies like to say that customers come for the price and stay for the quality.

The key to India's success in outsourcing has been the Capability Maturity Model and its successor, the CMMI, the process and organization assessment schemes set up by the US Department of Defense for evaluating suppliers.

I remember being surprised, a decade ago, when the first reports about CMM qualification came in. When the SEI introduced the five-level CMM, conventional wisdom held that no one in the foreseeable future would qualify at Level 3 or above. Very quickly, however, news of such qualifications appeared, with more than half coming from India. Why India? We have the answer now. For newcomers eager to establish their reputation with initially suspicious Western customers, such an independent outside certification proved a lifesaver.

I still hear comments that only low-level jobs can be outsourced and that all design will remain in the West. That's delusional.

The top Indian outsourcing companies operate at the same level as their Western counterparts, while certain countries like Russia specialize in advanced, high-level developments at or beyond the hardest in the West.

Observing that Intel was investing more than $1 billion in R&D in Bangalore and that J.P. Morgan would hire 4,500 new employees by 2007, the *New York Times* (5 December 2005) cited an analyst who stated that "This is way beyond mere cost savings; unless global banks are comfortable with the quality of work, they would not risk taking the work offshore."

In addition, an increasing portion of outsourcing in large companies, but also in some small ones, is internal: Companies establish their own branches or laboratories in places like Bangalore (ABB), Saint Petersburg (Sun), Beijing (Microsoft), and so on. The people there are as much a part of the company as those in the original locations. Everyone must compete on merit and results.

The frontier is moving. Eastern Europe and Ireland no longer have dirt-cheap prices; I talked recently to an outsourcing company in the Czech Republic whose developers—paid $5,000 a month—are becoming picky about the jobs they accept. In the ever-moving world of globalization, existing competitors will lose their advantages and new entrants will try their luck.

Not every country that wants to duplicate the Indian success will pull it

off. The recipe includes several ingredients:

- *Low salaries*. Success requires a university system that is both highly advanced and produces a large number of graduates in software—the two don't always go together as the case of Japan indicates.
- *Stable political system*. This ensures that customers needn't worry about tanks rolling in the streets next week or about paying bribes to corrupt officials.
- *Democracy*. Repressive regimes just don't foster the kind of independent, critical thinking that makes good software developers.
- *Open economy*. The social and business climate must encourage and reward entrepreneurship.
- *Expatriate network*. It also helps if, as in the Indian and Chinese cases, a sizable diaspora exists in the customer countries to provide the indispensable human contacts.

Not all the new contenders meet this full combination of criteria.

MAKING OFFSHORING WORK

What should programmers and other software professionals from the industrialized world do? While there is no need to panic, ignoring the offshoring phenomenon would be suicidal.

First, understand its impact on the software engineering process. For the past two years, Peter Kolb and I have been giving a course at ETH Zurich on software engineering for offshore development.

Designed for students and also held as an industry seminar, this course is apparently the first of its kind. Others surely will follow, and I hope that software engineering research too will start addressing the special circumstances of outsourced and offshore development.

To help in this process—and because the acronym was impossible to resist—we are organizing with Joseph Mathai, R&D head of Tata Consulting, the first international conference on Software Engineering Approaches for Offshore and Outsourced Development at ETH Zurich on 16-19 October 2006 (SEAFOOD); the purpose is to conduct an in-depth discussion of the technical issues raised by the offshoring phenomenon, as sketched in this article. The submission deadline is 15 April 2006 and the Call for Papers is at seafood.ethz.ch.

The development of offshoring also raises a new challenge for those of us entrusted with educating future software professionals in the industrialized world. If ever there was a justification for focusing computer science programs on teaching high-level system skills and analytical as well as synthetic reasoning, rather than just the technology du jour, this phenomenon provides it.

Not every country that wants to duplicate the Indian success will pull it off.

I have argued for such an approach before ("Software Engineering in the Academy," *Computer*, May 2001, pp. 28-35; and "The Inverted Curriculum in Practice," http://se.ethz.ch/~meyer/publications/teaching/sigcse2006.pdf). Now, more than ever, we have a duty to provide our students with solid long-term skills. It's not enough to give them the tools and languages that will get them a job on graduation day when legions of programmers, paid a fraction of Western salaries, have also mastered them.

Programming is, and will likely remain, a creative activity. The educational systems of democratic countries have a key advantage here: People who want to succeed in software development must possess—along with an ability to plan, schedule, and organize—the kind of inquisitive, critical mind that such systems encourage.

I don't think it's a coincidence that among the major Third World powers, the one that has succeeded so far in establishing a massive software offshore operation—India—also has the closest to a functioning democracy. To bolster their competitive advantage on the global software scene, Western universities must continue to provide a broadly based education that includes a strong scientific basis but does not exclude the humanities.

Along with education, software engineering research should take outsourcing into account. Existing software engineering principles and tools can help offshore projects, but there's far more to do. Most project management tools, for example, do not provide sophisticated enough support for distributed development. The same applies to configuration management.

In the area of process models, I hope we see significant improvement. CMMI and ISO 9000-series standards have obviously helped the industry organize itself, but their zeal to avoid technology commitments has limited their benefits. To put it less politely, a bit of substance wouldn't hurt. For example, am I the only one who, when teaching CMMI, has trouble hiding from the students that, however hard I try to look excited, I am hopelessly, desperately bored? What about a process model that wouldn't shy away from suggesting precise technology choices for, say, an object-oriented approach?

Another research area that should get a boost from offshore development's growth is requirements engineering. Maybe formal methods will get a second hearing from an industry that has often viewed them with outright hostility.

Unlike an English text that offers endless possibilities for subtle interpretive differences, a mathematical formula leaves no room for national or cultural nuances. Whether read in Palo Alto, Moscow, Krakow, or Sao Paulo, a given function is either total or partial, a given formula of predicate calculus either satisfiable or not, a given number either positive, negative, or zero.

The world of software construction will never be the same: We're just not in Kansas, or California, any more. If you are a software professional, you must adapt to the new reality and become the best at what you do by using the right tools and methods. Anything else is futile. Security by obscurity—the practice of producing code that no one else will understand, in the hope that the company will retain you out of desperation—won't protect you for long now.

What will make you truly irreplaceable is to show exceptional productivity and to deliver stable code—bug-free, extensible, reusable—that establishes your reputation as a truly irreplaceable resource. Also, make sure you know the business as well as the technology; that will set you apart from mere techies.

Cultivate system thinking and whole-life-cycle skills—there will always be a demand for people who master these, even at Western prices. Don't forget that tool investment, and using the top software technologies available, offers your best bet in this tough new world.

Finally, look for technical publications and professional events—including, I am sure, future ICSEs—that specifically address the impact of outsourcing and offshoring on the software engineering discipline.

Bertrand Meyer *is professor of software engineering at ETH Zurich and the chief architect of Eiffel Software in California. Contact him at Bertrand.Meyer@inf.ethz.ch.*

Editor: Neville Holmes, School of Computing, University of Tasmania; neville.holmes@utas.edu.au. Links to further material are at www.comp.utas.edu.au/users/nholmes/prfsn.