DOI: 10.1145/1721654.1721689

BY KEITH WRIGHT

Capstone Programming Courses Considered Harmful

WHEN EDGSTER DIJKSTRA WROTE HIS PAPER *Go To Statements Considered Harmful*,[3] programmers were lost in millions of lines of spaghetti code. Now programmers have lost their way again—this time amidst thousands of unread resumes. Between 2000 and 2004, the percentage of incoming computer science freshmen fell by 60%. Drop rates of 30-50% are common. Results are similar for the other computing related fields including Information Systems, Software Engineering, and Information Technology.[4] Many of these degree programs, which H.A. Simon referred to collectively as the Artificial Sciences, have failed or are about to fail.[12]

This loss of student interest in artificial sciences has been blamed in part on the impression that they require extraordinary programming skills. Computing journals perpetuate these notions. For example, in capstone (senior level) computer science courses, students should design and code a 'real world' application.[6] This article is an opposing viewpoint.

Software Development from 1975 to 2004 – An Insiders View

Before rejoining academia, I spent over 20 years as a professional programmer. Drawn to the field by my college-bred love for programming, I began my first job in 1974 when computers and compilers were unavailable except to academics, business owners, and employees. The principal education requirement for a professional programmer was a college degree that included programming courses. The essential qualities employers sought were analytical ability, intellectual curiosity, and loyalty. Programming was a rare skill, and programmers in short supply. My first job was with a startup company with a government contract to digitize highways and power lines—the beginnings of the technology found today in Google Earth, and Yahoo Maps.

Computer centers in those days were populated with machines such as IBM 360s, Burroughs 3700s, and UNIVAC 1100s. Along side were recent college under-graduates, experienced COBOL / FORTRAN programmers, and graduate students. I was part of a 15 person programmer team. My job was to write map plotter drivers. I was doing design and using math I'd just learned in college.

Because of the short supply, professional programmers in those days rarely stayed at one job more than a year; and I too soon moved on. Subcontracting with a Silicon Valley firm called Informatics, then I was part of a 100-person team developing a shop-floor control system for the United Airlines maintenance operations center in San Bruno, California. These kinds of projects were the beginnings of current supply-chain technology. Unfortunately when I arrived, the systems analysts hadn't yet completed their work. This meant there was nothing yet ready to program. That unfortunate situation was due to either poor planning, or an intentional effort to hire programmers before someone else did. In either case, United Airlines was billed for these wasted programmer hours.

After a few weeks I moved to a systems analyst job with a Virginia-based

company, Information Engineering Systems Corporation (IESC). There I was part of a large team doing data modeling for a supply chain project for the Meijer department stores data center in Grand Rapids, MI.[a] That job was fine until I found that the intended consumers of the data models – the programmers—were employed by a different subcontractor, Keane,[b] which operated under a separate service level agreement that allowed their programmers to proceed before the IESC data models could be developed. Again, wasted hours were billed to the customer. This bothered me. So I shortly moved on to my next job.

This time with Levi Strauss in San Francisco programming an interesting optimization algorithm to cut clothing patterns wasting the least fabric—part of an early just-in-time inventory system. This was a very enjoyable job, in which I used knowledge gained in college. Unfortunately, two years later, Levi Strauss off-shored their entire supply chain, and I was laid off.

My quest for rewarding ethical programming jobs continued for the next decade, taking me to Advanced Micro Devices, National Semiconductor, academia, and finally in 1998, IBM. My first job there was with a small Java team working on an interesting PC-based datamining application for insurance fraud detection. I was working with J2EE,[c] and learning Web technologies for the first time. But sadly for me, IBM, in response to the rapidly maturing business applications market, soon abandoned it.

By the year 2000, enterprise resource planning systems (ERP) had reduced the demand for custom business applications dramatically. ERP systems saw a large boost in sales in the 1990s as companies facing the Y2K problem replaced their systems with ERP. Prior to ERP, each organization department likely had their own custom information system. This architecture was expensive to maintain. Hence a competitive advantage accrued to those companies standardizing on a few standard ERP platforms, most notably now, SAP and Oracle.

a http://www.meijer.com

b http://www.keane.com

c Java 2 Enterprise Edition

IBM, wishing to maintain its share of the standard platform market, invested heavily in the Web infrastructure software market in the late 1990s; and by the year 2000, its WebSphere product line was a market leader. This line of middleware was millions of lines of code developed by teams distributed across the globe among hundreds of programmers. In 2000, I took a programming job on a WebSphere development team. By then, IBM programmers were known as *developers*. They were more technology wizards than programmers. They had to be technology wizards to set up a programming workstation. This tedious task involved installing a large stack of software including the J2EE, the Apache web server, the WebSphere Application Server, DB2, WebSphere Portal Server, and Tivoli Access Manager. This stack of (usually immature) software contained many intricate version dependencies.

There were few support people for the developers, because many of the pure UNIX and NT systems administrators had been laid off in the previous few years. As a result, most of the IBM developers did little programming. They spent their time doing systems administration, installing software, or testing it. The programming involved displaying and interfacing Web pages to the WebSphere Portal Server. This (side effect) programming required expert skill in J2EE, XML, and the complete line of WebSphere brand products. This type of programming was nothing like the functional programming I learned in college. It was similar to programming a DVD player.[2,5] To avoid this type of frustrating side effect programming, most IBM developers wanted eventually to become architects.

IBM architects had jobs that, although had little to do with programming, were more rewarding than those of developers. Architects lead the programming teams. This leadership included dispensing workstation images, test data, method signatures, and build paths. Having at least ten years with IBM, architects had power, and they did not always part with it willingly. Some of the preferred methods of retaining power included dispensing incorrect documentation, working odd schedules to avoid knowledge sharing, and resisting manager efforts to add new developers. IBM architects had well learned the hardest programming lesson: programmers who are dispensable will be eventually dispensed with. So, instead of ace programmers, IBM architects became experts in AIX, Solaris, HP-UX, Linux, DB2, Microsoft Windows, and the WebSphere products. Architects were highly sought, and highly paid. But they had little occasion to apply skills they learned in college.

My personal observations of the software industry are consistent with the ACM's 2006 report on software globalization.1 Page 36 of that report states, about programmers:

> *They are more likely to work on large software applications in teams that include applications specialists…they are expected to be masters of certain software platforms and interoperability standards…it will be increasingly important that….education enables the student …to be familiar with the tools and platforms that are increasingly standards in the international market place.*

Programming – Prospects for the Future

In summary, the author's industry experience from 1975 from 2004, and current economic literature suggests that, by 2004, a lucrative United States career as a generalist professional programmer was a thing of the past. And that was before many information technology services were substantially offshored.

The most widely cited estimate of the scale of this off-shoring is a 2002 Forrester projection that over the next 15 years, 3.3 million U.S. services industry jobs and $136 billion in wages will move offshore.[7] From 1997 to 2004, U.S. off-shoring of business, professional, and technical services increased 77%. There is little doubt that the growth of off-shoring has negatively affected both U.S. job availability and wages in the IT job market.[13]

According to the McKinsey Global Institute,[9] "a software developer who costs $60 an hour in the U.S. costs only $6 an hour in India." To take advantage of this comparative advantage, in 2004 IBM launched a $200 million Indian development center, to handle the bulk

of solutions development work for IBM worldwide.[8] IBM now has 15% of its global work force in India. Although U.S. firms sell 84% of the world's packaged software, according to the ACM's Software Globalization report, the U.S. employs only 50% of the related labor.[1]

According to a recent report released by the Brookings Institution, about 20% of the existing U.S. IT jobs will move to lower cost destinations by 2015.[d] The study predicted that about 60,000 jobs will be offshored between 2004 and 2015. Between March 2001 and November 2001 the IT industry shed 197,000 jobs. By March 2002 IT industry employment had declined by more than 270,000 jobs. Significant losses continued such that by March 2003, IT industry employment had fallen by an additional 113,000 jobs. Employment losses finally slowed by March 2004, although the industry still suffered a decline of 19,700 jobs. All told, these mounting losses meant that the IT industry lost 402,800 jobs between March 2001 and March 2004.

In spite of these gloomy statistics industry leaders and economic policy makers believe that the future of the American workforce is in information technology, working as what Robert Reich termed "symbolic analysts."[11] Supporting these claims are signs, since March of 2004, of a weak resurgence in the IT industry. The total number of available positions has increased almost 50% since April 2002, if Monster.com is a reasonable indicator.[13] The US Bureau of Labor Statistics forecasts 20%–50% job growth in all computing specialties by 2012, except computer operations, which is declining, and programming, which is flat.[10]

In summary, this article has described the dramatic decline in demand for U.S. programmers over the period from 1974 to 2004.[e] Let me conclude this section by saying that there is still some demand for US programmers. Most of this demand is for Web programming which now leads the total number of jobs requiring programming skills.[10] But the Web has radically changed the programming skill sets required. Web programming comprises a complex set of non-functional programming skills, not typically taught in college. As a result, many of today's college degree programs in artificial sciences need a complete redesign.

Recommendations

When designing computing curriculums, it is useful to first consider the capstone courses. Many of today's capstone courses involve locating real programming projects in the community.[6] However, this is often too time consuming. Furthermore, real projects are usually too difficult for college seniors. Other capstone computing courses have students working individually on imaginary projects, giving them a chance to build something they own. Unfortunately, this may give students a poor understanding of the real world. Finally, we do students a disservice if capstone courses give them the impression they are likely to become professional programmers.

It is important to remember that, in the ACM Education Board's Great Principles of Computing project, programming is only one of four core practices. The other three are systems thinking, modeling, and innovating. Deemphasizing programming and giving systems thinking a more prominent role in artificial science curricula would improve its appeal. (For an excellent discussion of creative ideas for adding innovation to Computer Science curricula we suggest.[4])

Research shows that students find bioinformatics and molecular biology more attractive than the artificial sciences.[4] So, why not look to the medical school curriculums for guidance for designing curricula for artificial science programs? The similarities of the medical profession to today's IT profession are mentioned in the ACM report.[1] That report recommends that student training in artificial sciences needs to be much broader, and deeper in certain areas. Medical schools provide that depth in part by arranging, instead of capstone courses, internships in the various major specialties. These internships extend for at least two years in most schools.

What are appropriate major specialties of artificial science? I believe part of the answer lies in industry best practices like those captured in the IT Infrastructure Library (ITIL) and the corresponding international standard - ISO 20000.[f, g] So I suggest, to designers of future artificial science curriculums, that capstone programming projects be replaced with the following internships:

- *Service desk.* Here students would serve as assistants in large data center service desks. Students could handle routine first line support. Especially incident recording and escalation.
- *Change management.* Here students would work with programmers and change control specialists. Program maintenance should be stressed here.
- *Programming.* In this rotation, students would work along side programming teams developing new systems. Requirements analysis, tool usage, version control, and system design would be strongly emphasized.
- *Networking.* Students here would assist in network configuration management, and diagnosing traffic problems.
- *Problem Control.* Here students would learn operating system administration.
- *Service level management.* Here an interdisciplinary approach is required to understand how services should be conceived, designed, delivered, and supported. Students could work as executive assistants to first line IT managers. Emphasized would be the writing of ethical service level agreements, underpinning contracts, and operational level agreements.

These internships correspond to process areas for IT service management which are treated at length in ITIL. This best practice framework mentions five other such process areas, but I do not mention them here for the sake of brevity.

Conclusion

In summary, enrollment in U.S. university computer related degree programs has plummeted in the past decade. Yet the field of information technology remains one of the stronger careers for those fortunate enough to receive the right training. This article a dilemma for many of today's U.S. universities. This article offered perspective and sugges-

d *Silicon Valley to Offshore Majority of Jobs Over 10 Yrs - Brookings*, Global Sourcing NOW, 2/13/2007

e Demand for database, ERP (such as SAP and Oracle / People Soft), and ecommerce servers skills (such as WebSphere and WebLogic) are on the rise.[10]

f http://www.itil.co.uk

g http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=41332&scopelist=PROGRAMME

tions to universities grappling with this dilemma and presented ideas on how to redesign computer curriculums to address modern economic needs. These recommendations included replacing programming capstone courses with internships that would supply depth in standard platform training, (for example, UNIX, SAP, Oracle) business processes, and IT service management. C

References

1. ACM (Association of Computing Machinery) Globalization and Offshoring of Software. *ACM Job Migration Task Force*; http://www.acm.org/globalizationreport/. 2006.
2. Brooks, F.P. *No Silver Bullet: Essence and Accidents of Software Engineering. Computer, 20*, 4 (April 1987).
3. Dijkstra, E.W.G. *Go To statement considered harmful. Comm. ACM 11*, 3 (Mar.1968).
4. Denning, P.J. and McGettrick. *Re-centering computer science. Comm. ACM 48*, 11, (Nov. 2005).
5. Hudak P. Conception, evolution and application of functional programming languages. *ACM Computing Surveys 21*, 3, (Sept. 1989).
6. Martin, F. Toy projects considered harmful. *Comm. ACM 49*, 7 (July 2006).
7. McCarthy, John C. *3.3 Million U.S. Services Jobs to go Offshore*. Forrester, Nov. 11, 2002; http://www.forrester.com/ER/Research/Brief/Excerpt/0,1317,15900,FF.html.
8. McDougall, P.C. IBM to move all solutions development operations to India. *InformationWeek*, (Mar 8, 2006).
9. McKinsey Global Institute. *Offshoring: Is it a win-win game?* San Francisco: McKinsey & Company. 2003.
10. Prabhakar ,B. and Litecky, C. R. IT skills in a tough job market. *Comm. ACM 48*, 10, (Oct. 2005).
11. Reich, R. *The work of nations*. NY: Alfred A. Knopf. 1991.
12. Simon, H. A. *The Sciences of the Artificial*. The MIT Press; 3 edition. (Oct. 1, 1996).
13. Srivastav S. and Theodore, N. A long jobless recovery: Information technology labor markets after the bursting of the high-tech Bubble. *The Journal of Labor and Society 8*, (Mar. 2005), 1089-7011.

Dr. Wright (WrightM@uhd.edu) is currently teaching business intelligence courses at the University of Houston-Downtown, TX.