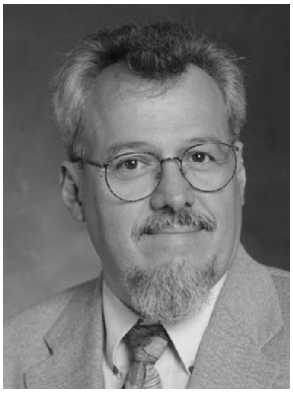
Improving software practice through education:

Challenges and future trends

Timothy C. Lethbridge, Jorge Díaz-Herrera, Richard J. LeBlanc Jr. and J. Barrie Thompson

Timothy C. Lethbridge is a Professor of Software Engineering and Computer Science at the University of Ottawa, Canada. His research interests include software engineering tools, software modeling, knowledge management and software engineering education. He is steering committee chair of the Conference on Software Engineering Education and Training (CSEE&T), a member of the IEEE-CS committee for the Certified Software Development Professional, and a member of the board of the Computer Science Accreditation Council in Canada. He was curriculum co-chair of SE-2004 and is co-author of a McGraw-Hill textbook on software engineering.



Jorge L. Díaz-Herrera is Professor of Computer Science and founding Dean of the B. Thomas Golisano College of Computing and Information Sciences at the Rochester Institute of Technology in Rochester, New York. Prior to this appointment, he held several faculty and administrative positions in the USA, including close to five years at the Software Engineering Institute teaching in Carnegie Mellon's Master of Software Engineering program. He was co-editor of SE2004.

Richard J. LeBlanc, Jr. is a Professor in the College of Computing at the Georgia Institute of Technology. He was co-chair of SE2004 Steering Committee. He has served as Chair and Vice Chair of the ACM Education Board and is currently a member of the ACM Education Council. He is also a member of IFIP Working Group 3.2 (Informatics and ICT in Higher Education). His current interests include computing education and software engineering.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

IEEE COMPUTER SOCIETY

J. Barrie Thompson is a National Teaching Fellow in the UK and is Professor of Applied Software Engineering in the School of Computing and Technology at the University of Sunderland, UK. His prime interests are educational, professional and ethical issues associated with software engineering. He promotes the development of innovative teaching approaches relevant to the needs of industry. He was a member of the SE2004 steering committee, and since January 2005 has been chair of IFIP Working Group 3.4, Professional and Vocational Education and Training.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

IEEE COMPUTER SOCIETY

Improving software practice through education: Challenges and future trends

Timothy C. Lethbridge
University of Ottawa, Canada
tcl@site.uottawa.ca

Jorge Díaz-Herrera
Rochester Institute of Technology, USA
jdiaz@gccis.rit.edu

Richard J. LeBlanc Jr.
Georgia Tech, USA
rich@cc.gatech.edu

J. Barrie Thompson
University of Sunderland, UK
barrie.thompson@sunderland.ac.uk

Abstract

We argue that the software engineering (SE) community could have a significant impact on the future of the discipline by focusing its efforts on improving the education of software engineers. There are some bright spots such as the various projects to codify knowledge, and the development of undergraduate SE programs. However, there remain several key challenges, each of which is addressed in this paper: The challenges are 1) making programs attractive to students, 2) focusing education appropriately, 3) communicating industrial reality more effectively, 4) defining curricula that are forward-looking, 5) providing education for existing practitioners, 6) making SE education more evidence-based, 7) ensuring that SE educators have the necessary background, and 8) raising the prestige and quality of SE educational research. For each challenge, we provide action items and open research questions.

1. Introduction

In almost any field of human endeavor, advancement occurs at least as much through the education of practitioners as it does by the development of new technologies, tools and techniques. One reason for this is that if practitioners lack knowledge of what researchers have already discovered, they are in no position to absorb new findings from researchers. A second reason is that a highly knowledgeable practitioner community will foster a general climate of respect for knowledge and best practices.

The above is evidenced in fields such as medicine, where advanced and ongoing education is mandatory. Medical schools put considerable emphasis on educational innovation, pioneering such approaches as Problem Based Learning [1], virtual dissection [2] and short sequential learning modules, as opposed to parallel courses. Furthermore, research advances rapidly find their way into the curriculum by necessity: Not to apply recently-developed diagnostic or surgical techniques may constitute malpractice.

The situation in software engineering (SE) is radically different. Firstly, as Figure 1 shows, only 40% of computer industry workers actually have a computing-related education. And most of this 40% are not educated in key portions of the SE body of knowledge, such as requirements, architecture, testing, human factors and project management. Most practitioners are merely skilled at programming in a few popular languages, as well as using specific technology products such as database management and web development tools.

Due to the relatively low dissemination of the broad body of SE knowledge, very little of the advanced knowledge developed by the research community, such as that contained in the papers presented at ICSE conferences, will have impact on practitioners. In addition, most practitioners have little awareness of what they don't know, have even less of a sense of the potential value in learning it, have no time to upgrade their knowledge anyway, and have few incentives to push them in the right direction. Indeed, as Fred Brooks put it: "In no other discipline is the gulf between best practice and typical practice so wide." [3]

The state of affairs in many other branches of engineering is intermediate between that in medicine and that in SE. Reasons for this include the much longer period of development of the other branches, which has allowed for the evolution of more stable bodies of knowledge, the development of

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

IEEE COMPUTER SOCIETY

licensing/certification procedures, and the fact that most practitioners are actually graduates from accredited programs that taught those bodies of knowledge typically required for professional certification and registration.

Nevertheless, the situation in SE is not completely dismal. Efforts such as SWEBOK [4], SE2004 [5] and CSDP [6], as well as the establishment, throughout the world, of accredited undergraduate programs and numerous graduate programs – including some Ph.D. programs – are seeding industry and academia with increasing numbers of real SE experts who will, we hope, gradually raise the level of professionalism.

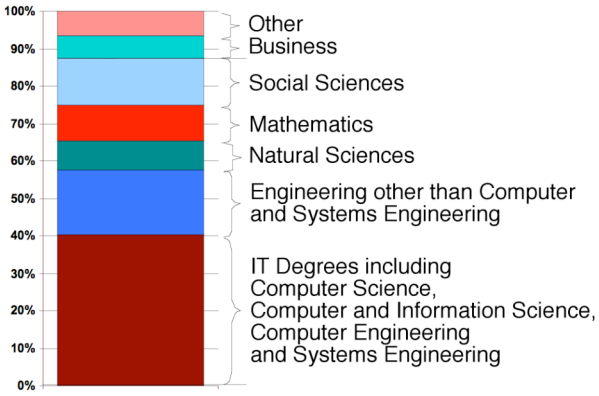


Figure 1: Educational background of professional-level IT workers (adapted from [7])

In this paper we will therefore attempt to show that, as a community, the SE knowledge-leaders epitomized by ICSE attendees can do more to advance software engineering by focusing on improving education and training than by any other activity.

In the next section, we survey recent trends; then in Section 3, we will present eight challenges the software engineering community needs to confront in order to advance the education of SE practitioners.

2. Developments that have advanced the software engineering profession

Software engineering education has developed as the profession itself has matured over the last 38 years. Tomayko provided a history of the first 30 years of SE education in a landmark 1998 article [8]. As that article was being written, SE education was rapidly blossoming, as will become clear from reading the remainder of this section.

Historically, SE developed as a sub-discipline of computer science (CS), so among the first manifestations of SE education were courses within CS programs. Several authors wrote textbooks that became widely used in such courses; most notable are the textbooks of Sommerville [9] and Pressman [10], which were published in their first editions in the 1980's and are still widely used today, after having passed through many editions.

As the software engineering profession developed further, Master's programs appeared (See section 2.2). However, it was not until the late 1990's that the explosion of progress in both software engineering professionalism and education occurred [11]. Key events and activities characterizing this explosion were:

- The establishment of the Software Engineering Code of Ethics [12].
- The announcement by the State of Texas that it would license software engineers beginning in 1999 [13], and similar moves by a few other jurisdictions.
- The establishment of undergraduate programs, which had initially appeared in Australia, but now started rapidly appearing in Europe, Canada and the U.S. The establishment of Ph.D. programs followed.
- The adoption of accreditation criteria in the U.S. for educational programs in software engineering. ABET accepted the final set of criteria in 1999 [14].
- The development of SWEBOK and SE2004, discussed in the next section.
- The development of the IEEE Certified Software Development Professional designation [6].

In what follows we discuss some of these developments.

2.1. Categorization of knowledge and positioning of SE as a computing discipline

Until the late 1990's, exactly what constituted the field of software engineering, and hence what should be taught in educational programs, was left in the hands of individual educators and textbook authors. Although this resulted in a healthy diversity of courses and programs, there was a strong need to develop a well-accepted consensus understanding of the field to serve as a guide for all forms of educational efforts.

One effort that led in this direction was a survey one of the authors of this paper performed in 1998 [15, 16] in which over 200 software professionals were asked several questions about 75 topics taught in computing programs. They were asked how much they had learned in university about each topic, how much they know now, and how important the topic has been in their work. From this it was possible to identify not only the topics that seemed to be most relevant, but also those topics where education was most lacking.

2.1.1. SWEBOK. Probably the best-known knowledge cataloging project is SWEBOK [4, 17]. This divides

IEEE COMPUTER SOCIETY

SE into a set of knowledge areas, and outlines the generally accepted topics in those areas.

SWEBOK was developed in a series of iterations over several years, culminating in the publication of its "Ironman" version in 2004. Development of SWEBOK was characterized by detailed attention to its review process: Large numbers of people from around the world provided feedback on each iteration, and the editors responded to all comments publicly as each subsequent iteration was developed.

The resulting document is not without its detractors, who complain about such things as missing topics, or a structure they find suboptimal. However, SWEBOK's key strengths are that it is widely accepted as an excellent baseline, and it is cited by many curriculum developers as the root of where their outline of the field could came from. It became an international standard through (ISO/IEC JTC1/SC7) [18].

Efforts are underway to create the next version of SWEBOK. It will be harmonized with SE2004 and CSDP, both of which we will discuss shortly.

2.1.2. The Computing Curricula Project. Overlapping with the development of SWEBOK, the ACM and the IEEE-CS sponsored a cooperative project to define curricula for the different kinds of undergraduate degree programs in computing [19]. As of 2006, there are five curriculum volumes covering computer science, information systems, software engineering (SE2004 discussed below), computer engineering, and information technology.

A sixth volume, the Overview Report, defines shared characteristics of all computing programs (Section 3.6 of [19]) and the differences between software engineering and its sister computing professions (Tables 3.1 through 3.3 of [19]). This enables those establishing a software engineering educational program to position it well.

As an example of how the Overview Report helps situate software engineering, Figure 2, reproduced from [19], shows how knowledge can be positioned on two dimensions, from theoretical to applied (x-axis) and from hardware to business/organizations (y-axis). The ellipse in the figure shows how software engineering covers knowledge at the center of this range. Other figures in [19] show that the sister disciplines focus on other parts of the plane.

2.1.3. SE2004. To understand what should constitute a software engineering program in much greater depth, the place to look is *Software Engineering 2004* (SE2004) [5, 20, 21], one of the volumes of the IEEE/ACM computing curricula effort discussed above.

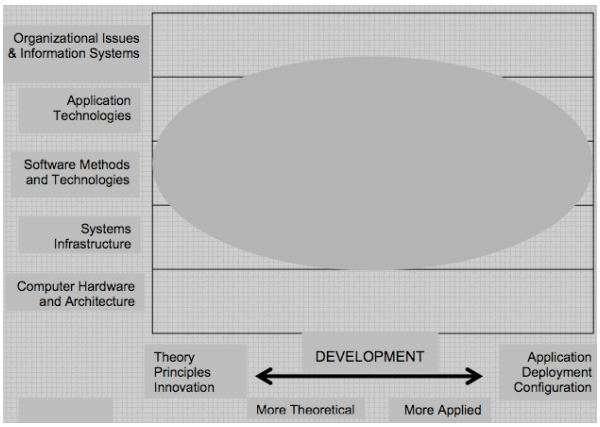


Figure 2: SE in the computing spectrum (from [19]).

Whereas SWEBOK is designed to describe the generally-accepted SE knowledge that should be known by a practicing professional, SE2004 describes what needs to be taught in an undergraduate program. This means SE2004 has broader coverage since it includes necessary aspects of computer science and mathematics, as well as general engineering principles. The SE2004 developers started with an early version of SWEBOK as a baseline, but as SE2004 progressed, its content diverged from SWEBOK. This is partly because it needed to cover a broader scope, and partly due to feedback from numerous reviewers. As with SWEBOK, the comments of all of reviewers are publicly available on the SE2004 website, as are the committee's responses to each comment.

The knowledge component of SE2004 is commonly referred to by its chapter title as "Software Engineering Education Knowledge" (SEEK). SEEK is structured as Knowledge Areas (KA), broken down into units and topics. (See Table 1 for a brief overview). Each topic is given a Blooms Taxonomy level of cognitive complexity indicating the level of understanding that should be achieved. The units are given a number of "hours," which helps guide educators to allocate portions of the curriculum to each unit.

The list of SEEK KAs presented in Table 1 has an interesting resemblance to the topics in this Frontiers of Software Engineering volume. The authors of several papers, in fact, emphasize the need to improve education. For example Dyb et al [22] discuss the need to educate students in empirical methods (in SE2004's FND KA), while Bertolino et al discuss the need for more education in Testing [23] (VAV), and Taylor and van der Hoek [24] discuss education in Design (DES).

In addition to the SEEK, SE2004 also contains model courses with suggested SEEK coverage, model curricula (suggestions regarding how courses can be grouped), educational outcomes, and a set of general guidelines to help curriculum designers and educators.

IEEE COMPUTER SOCIETY

Table 1: The Knowledge Areas of SEEK

CMP	**Computing Essentials** (172h): computer science topics, and construction technologies.
FND	**Mathematical & Engineering Fundamentals** (89h): discrete math, statistics, measurement, and economics.
MAA	**Modeling & Analysis** (53h): general modeling notations, and requirements.
PRF	**Professional Practice** (35h): group dynamics, psychology, communication skills, professionalism and ethics.
DES	**Software Design** (45h): concepts, strategies, architecture and human-computer interface design.
VAV	**Software Verification and Validation** (42h): reviews, testing, human-computer interface testing and evaluation, as well as problem analysis.
EVL	**Software Evolution** (10h): processes and activities.
PRO	**Software Process** (13h): concepts, implementation
QUA	**Software Quality** (16h): quality concepts, culture, standards and processes.
MGT	**Software Management** (19h): planning, personnel, control and configuration management

Many interesting issues arose while developing SE2004. For example, groups of people wanted deeper emphasis on areas such as formal methods or process. The end result is intended to be balanced among such competing perspectives, yet it leaves plenty of space to tailor a curriculum for deeper coverage of desired areas. Discussion of the controversies encountered by SE2004's developers can be found in [21].

SE2004's development and evaluation approach, the tools used to support the work, and the level of international participation in the effort were formally recognized in the 2004 annual report of the ACM Education Board as setting standards for the development practices for future ACM-sponsored curriculum guidelines [25].

2.1.4. Defining the knowledge base: Future trends. SWEBOK and the Computing Curricula will continue to evolve and improve. Undoubtedly as they do so, the profession will become better defined. The key will be to ensure that the knowledge captured does not stagnate – this issue will be revisited in Section 3.4

2.2. The growth of university SE programs

In the late 1970's, an IEEE–CS curriculum project stimulated the creation of several software engineering Master's programs in the U.S. [26, 27]. While this effort centered on graduate education, it formed the basis for software engineering education in general.

There is a minority body of opinion that software engineering should remain a specialty at the Master's level, and that undergraduates should simply study computer science. In fact, Mary Shaw specifically advised this in her ICSE 2000 Future of Software Engineering paper [28]. Nevertheless, undergraduate SE programs have blossomed; in the next three subsections we will briefly overview the situation in three countries.[1]

2.2.1. Undergraduate programs in the U.S. Bachelors SE programs in the USA are steadily increasing in number. They started to appear in 1996 [29, 30, 31, 32], and since then, over 30 programs have come into existence, and several more are under development [33].

Engineering and Technology programs in the U.S., including CS, are accredited by ABET. The first such school visits for SE programs by accreditation teams started in the fall of 2002, as pilots to "test" new accreditation criteria. The first schools were fully accredited in the summer of 2005.

2.2.2. Undergraduate programs in Canada. In 1997, the first SE programs were established in Canada at the University of Ottawa, McMaster University and the University of Western Ontario. Since then, most large universities with CS or CE programs have created SE programs as well, so the number of SE programs now exceeds 20.

Two organizations currently accredit SE programs. The Canadian Engineering Accreditation Board (CEAB) accredits all engineering programs, while the Computer Science Accreditation Council (CSAC) accredits computing programs, using specific guidelines for software engineering. As of 2006, CEAB lists eight accredited programs [34], while CSAC lists three, as well as numerous SE options in CS programs [35]. Some programs are accredited by both bodies.

There has been controversy in Canada arising from the use of the term Software Engineering by a program not developed according to principles established by the CEAB. An attempt was made to establish a joint accreditation protocol between the CEAB and CSAC, but negotiations have stalled.

2.2.3. Programs in the UK. In the UK, the first complete named software engineering programs began at Imperial College in 1985 and at the University of Sheffield in 1988 [36, 37]. By the end of that decade, recommendations had been produced by the British Computer Society and the Institution of Electrical

[1] It should be noted that SE2004 provides explicit guidance to develop undergraduate programs in a number of locales such as Japan, North America and the UK. See Section 2.1.3.

IEEE COMPUTER SOCIETY

Engineers regarding undergraduate curricula for software engineering [38].

The number of SE programs at certificate, diploma, undergraduate, and master's levels increased steadily during the 1990's; at the time of student applications for the 2002-03 academic year there were at least 55 different single subject undergraduate level programs being offered in SE [39].

In the UK, the government ensures standards via the Quality Assurance Agency for Higher Education (known as the QAA). This publishes a Framework for Higher Education Qualifications, and Subject Benchmarks [40]. The framework defines levels of qualifications (certificate, diploma, degree, etc.). The computing benchmark document recognizes the diversity of computing programs within the UK and the fact that in 1999 there were over 2,400 undergraduate programs in computing.

Standards are also ensured by the external examiner system and by accreditation. Most universities in the UK operate an external examiner system to ensure standards. A normal role for external examiners is to ensure that students are treated fairly, and that standards are maintained at a level comparable with universities elsewhere in the UK.

Program accreditation visits by panels from the British Computer Society or the Institution of Engineering and Technology (formerly called the Institution of Electrical Engineers) offer another means of assessing standards within UK computing programs including those that address software engineering.

2.2.4. Ph.D. programs. It is still the norm in many countries for new software engineering PhD to study in programs that are labeled 'Computer Science' or 'Computer Engineering.' There are, however, a few U.S. universities that have now established software engineering Ph.D. programs [41, 42]. These include Carnegie Mellon University, Naval Postgraduate School, North Dakota State University, and the University of Texas at Dallas. This is yet another indicator that the profession of software engineering is becoming more distinct from its sister disciplines.

2.2.5. Educational programs: Future trends. We anticipate that an ever-increasing fraction of professionals who develop software will adopt the title software engineer.

It also seems clear that the title 'software engineering' will become as common as the terms 'computer science' and 'computer engineering' in degree titles. This is already the case in the UK and Canada and is becoming so in the U.S. in institutions that offer engineering programs. Using the term 'software engineering' in a degree title helps guide the consumer to understand that the holder of the degree would have learned the type of material found in SWEBOK and/or SE2004, which in turn is clearly material in great demand in industry.

Society appears to be ready to call all software developers "software engineers" regardless of their degree title: The press and film credits use the term "software engineers" predominantly now.

The above, however, raises a concern for the long-term viability of computer science programs. We clearly still need computer science graduates, but perhaps in smaller numbers than we need software engineers, much in the same way that we need physicists but in smaller numbers than electrical or mechanical engineers. There are clear projections in the job market in the U.S. that support this claim: "Computer software engineers" are projected to be one of the fastest growing occupations over the 2004-2014 period with a growth close to 50% [43]. However, the challenge will be to maintain critical mass in CS programs as software engineering gradually comes to predominate.

2.3 Certification and Licensing

Approaches to certify individuals vary from country to country. To establish a basis for certification at the international level, the IEEE Computer Society has made available worldwide the Certified Software Development Professional (CSDP) designation [6]. To achieve this designation you have to have a university degree, to have 9000 hours of software engineering work experience, and to pass an exam. The IEEE originally intended to use the term 'Software Engineering' for this certification but were unable to do so since this conflicted with laws in many jurisdictions reserving the term 'engineer' for someone granted an engineering license.

That approach, obtaining an engineering license, is now available to software engineers in some US states, and all Canadian provinces. In fact, in Canada, a considerable number of people have been recently licensed under software engineering criteria. The CEAB rules strongly encourage all software engineering professors to obtain their licenses.

The situation in the UK is that the British Computer Society and The Institution of Engineering and Technology are both members of the Engineering Council, a professional body that represents all the engineering professions. As such, each can offer Chartered status to suitably qualified and experienced members. Those members in the software engineering field would normally apply to be a Chartered Engineer (C.Eng.); however, some may see it as equally advantageous to be designated as a Chartered

IEEE COMPUTER SOCIETY

Information Technology Professional (C.I.T.P.). The latter is a newer professional designation that both bodies are promoting, especially the BCS through its Professionalism in IT Programme [44].

It should be noted that Canada also has a similar IT professional certification, the Information Systems Professional (ISP). Software engineers may choose to become an ISP, following criteria similar to those involved in becoming a professional engineer. The difference is that the ISP does not confer a license. Theoretically, on the other hand, the engineering societies in Canada could start to object to people performing software engineering (as defined perhaps by a court) who do not have a P.Eng. Whether that will ever happen is a cause of debate and concern for some.

2.3.1. Certification and licensing: Future trends. In the UK and many European countries, people performing software development can become Chartered Engineers regardless of whether they have graduated from a CS or SE program. This seems reasonable, given that in industry both CS and SE graduates software "engineers."

In North America, especially Canada, more and more recent graduates of SE programs are in a position to obtain their P.Eng status. However, our perspective is that most will not, simply because it is an expensive task that is currently not required by most employers. Most electrical and computer engineers do not obtain their P.Eng. either.

Our expectation, therefore, is that the certification and licensing situation will only very slowly evolve in North America. Perhaps the CSDP has the greatest opportunity to make inroads, because employers can clearly see that if someone has their CSDP, then they have a depth of knowledge of software engineering.

To further capitalize on the opportunities CSDP provides, the IEEE Computer Society is developing an "entry level" CSDP for recent graduates. This will serve as a stepping stone to full CSDP certification.

2.4 Educational materials and delivery methods

Textbooks, discussed earlier, as well as journals and conference proceedings in traditional libraries once served as the key reference sources for students and educators. However, these resources have now been supplanted to a considerable extent by electronic sources such as the IEEE and ACM digital libraries. There are obviously countless other websites for students to refer to on technical topics. The following, however, are some of the most notable sites for educators.

The Institution of Engineering and Technology in the UK has started placing high-quality lectures online [45]. As their own website says, "IET.tv has become the largest repository of interactive engineering and technical presentations on the Internet."

An interesting resource to support teaching of software engineering ethics through case studies is the Online Ethics web site [46].

SWENET [47] is an informal organization attempting to build lecture materials in the form of reusable modules. The objective of SWENET is to serve as a repository of materials shared among many educators, yet subject to quality assurance.

Finally, Seidman and Naveda [48] have recently published a book targeted at helping people pass the CSDP examination.

2.4.1. Materials and Delivery: Future trends. The main trends we see in SE course delivery are:

- A move towards greater use of shared resources such as IET.tv, SWENET modules, and other material found on the web. We anticipate that the developers of these materials will tag them with the relevant SE2004 and SWEBOK topics, allowing course instructors to pull together a rich compendium of resources relevant to what he or she plans to teach.
- Gradual adoption of a variety of innovative approaches to instilling a deeper practical appreciation of the material in students. How to manage group projects effectively has been a dominant theme in many recent ICSE education tracks, and sessions of the Conference on Software Education and Training (CSEE&T) [49]. We expect to see SE practitioners adopting approaches that reduce the amount of lecturing: These include studio and Problem Based Learning approaches.
- Adoption of improved instructional tools, such as simulations, computer-based training, etc.

2.5 Summary

In this section we have highlighted key developments in software engineering education and professionalism. In the next section we look at challenges for the future.

3. The challenges

In this section we discuss eight challenges on which we believe efforts should be focused in the future so as to improve not only the quality of software engineering education, but also the quality of the workforce, and, consequently, the quality of software developed.

IEEE COMPUTER SOCIETY

The reader might be interested in comparing the challenges listed here to the challenges in computing education outlined in [50]. For instance, Grand Challenge 1 "Perception of computing" is closely related to the challenge outlined in Section 3.1 below.

3.1 Making programs attractive to appeal to good students and meet societal demand

Everyone teaching a computing discipline in a university in a Western country is acutely aware of the enrollment declines that have occurred since 2000. In a recent workshop sponsored by the U.S. National Science Foundation, computing educators and industry representatives debated this issue at length [51]. They identified and discussed necessary transformations in undergraduate computing to address the problem.

Many reasons have been posited for the declines, including: public fear following the dot-com bust and the rise of offshoring; young people so immersed in computers that they do not see the excitement to them any more [51]; perceptions that the field offers little to those who want to 'contribute to society'; and the stereotype of the 'nerd' coding all night with no social contact. The latter two particularly seem to differentially turn away women, making the field increasingly male dominated – in a vicious-cycle manner. All of these reasons, although partially founded in fact, are largely false impressions; however it is impressions that count, so they must be countered.

No matter what the causes, the decline threatens the discipline on several fronts: Bright minds are going into other disciplines, and low enrollments result in low levels of hiring of new Ph.D.'s which will restrict the capacity for research in the field.

The declines impact all fields of computing, not just software engineering. Unfortunately, some aspects of the public perception might impact software engineering even more than computer science: One such impact is that the public now uses the term 'software engineer' ubiquitously to refer to all software developers (as discussed earlier in this paper). The negative stereotypes tend to focus particularly hard on SE. Secondly, many of the bright mathematical minds interested in computing tend to be attracted to CS due to its more deterministic and mathematical problems in areas such as algorithms, artificial intelligence, etc. Thirdly, the somewhat-true perception that SE is much about process acts as a detractor: We have witnessed students suddenly being put off when they learn the importance of process; it is as though the field has the less-glamorous aspects of business school, without the cachet and the money to be earned.

Ironically, software engineering has also been called the "best profession" by Money Magazine [52], and significant shortages of software developers have been forecast for the near future [43].

Perceptions or not, this is reality, so how can we make software engineering more attractive to incoming university students? How do we transform the negative perceptions to positive facts? The ACM and the Computing Research Association [53] have put significant effort into public awareness campaigns for computing in general, but public awareness campaigns at the institutional level should also be implemented. Other strategies include implementing educational innovations, such as those recommended in [51], and countering the negative stereotypes on all possible occasions, as discussed below. Nevertheless, it is important to point out that the threat of offshoring of software development is real [54], and this needs to be presented in the appropriate context.

3.1.1. Action items for the community: One of the most important things needed is for SE practitioners and academics to apprise themselves of the positive aspects of the profession, and make sure these are communicated to K-12 educators, guidance counselors, university recruiters, and more directly, to high school students and incoming freshman. Parents and guardians also do not have the appropriate information to encourage and support their children's potential inclination into computing programs, especially software engineering.

Some of the positive factors, directly countering the negative perceptions described above include:

- There is a good job market, with shortages expected.
- Society has an inexhaustible appetite for software, particularly in the entertainment and gaming industry.
- Offshoring is not having a significant impact since the field is growing faster than offshoring, and since many jobs require close contact with customers [55].
- There are exciting problems to tackle both in terms of systems to be developed, and research problems for those with advanced degrees [56].
- The stereotype of long nights of coding is only a result of bad management in some companies. New trends countering this stereotype include model-driven development, which is making design more visual and abstract, as well as TSP, PSP and the 'agile' approaches, all of which use careful management to avoid the need for overtime.
- When SE is practiced well, social contact can be quite high: Teams work together intensely, and interaction with users and customers is essential and frequent.
- Many SE projects have a huge and positive impact on society, particularly in the medical domain.

IEEE COMPUTER SOCIETY

Another action item is that we should work towards broadening and deepening CS/SE education in high schools, so that it is not just 'programming' or 'history of computers.'

3.1.2. Research questions: The following are some of the open questions in this area. For each, there has been some research and speculation, but deeper research is needed.

- To what extent do software engineering work practices match the negative stereotypes, or do they match the more positive environment we have described above?
- What is the importance of each of the factors in helping high-school students and their parents decide for or against studying software engineering?
- What are the inherent limits on offshoring imposed by the need to work with users? What is the impact for national security and defense applications?
- How inexhaustible is the demand for software really likely to be, and where are new sources of demand likely to come from? It would seem that if software engineers really did achieve the utopia of flexible, reusable libraries for rapidly developing applications, then they would do themselves out of a job. The market for new operating systems and programming languages is certainly minimal, but a better quantification of the ongoing needs for other types of software would not only help convince new students to enter the field, but would also help educators focus their efforts.
- Recognizing the famous productivity gap between the top and bottom developers [57], what would be the effect of an educational strategy that sought to only teach the elites, e.g. with aptitude tests such as those required to enter medical or law school? Might it be possible to secure industrial funding for this, since such students would be in high demand?

3.2 Understanding the dimensions of the field so we can focus education appropriately

As a profession matures it necessarily divides into specialties. The computing profession as a whole, for example, only recently finished the task of defining separate program recommendations for each of the five areas in the Computing Curricula project discussed in Section 2.1.

Software engineering also has its specialties. As a priority, we need to learn more about these specialties in terms of how practitioners work and what their different educational needs are. This will allow for: a) better-tailored programs, and b) a smaller core.

We are currently witnessing a movement by several schools in the U.S. to "diversify" computing education by connecting it to other "cognate" disciplines [58, 59]. There are also SE Bachelor's programs with specific specialization domains [60].

3.2.1. Research questions:

- What are the different educational needs of web site developers, games developers, software engineers who aspire to be IT managers, real-time systems developers, etc?
- What aspects of a software engineer's education should be moved to earlier years? Historically several topics have moved in this direction; for example programming has been moving to the high school level to some extent, and object-oriented techniques moved from being an advanced topic to the first year in the 1990's.
- What aspects should be moved to higher years? The SE2004 guidelines suggest that process and management topics are difficult for students to grasp unless they have had on-the-job experience. To what extent is this true, and what aspects of these topics can nevertheless be taught in earlier years?
- What aspects could be removed from the core in certain contexts? Calculus for example? Although this raises the ire of some, it has been suggested that other equally-rigorous types of mathematics such as deeper statistics could replace calculus.
- What is the cost of omitting the teaching of topics such as mainframes and Cobol maintenance? It has been argued [61] that the failure of universities to teach these topics will make skills shortages particularly severe, and will also result in quality problems and increased costs for industry as the base of experts diminishes.
- How should key industry trends such as the trend towards component-based engineering influence education?
- What are the benefits and costs of software engineering curricula that are: 'design centric,' 'process centric,' 'testing centric,' 'formal methods centric,' etc.?
- What about the trends towards application domain knowledge? Arguments are being made for introducing cross-disciplinary material in the curriculum by combining computing with *cognates* to form interdisciplinary educational programs [51]. Or, should this type of specialization be left for graduate level education?

IEEE COMPUTER SOCIETY

3.3 Communicating real-world industrial practices more effectively to students

Anyone who has tried to teach topics such as ethics, quality, process, configuration management, maintenance and requirements will recognize the glassy-eyed appearance in the eyes of some (or most) students. These are critical topics for industrial practice, yet it is a particular challenge to motivate students to feel passionate in these areas, and hence learn what they need to know.

The 14th Curriculum Guideline in SE2004 [5] points out that the curriculum should have a real-world basis, but the question is how to achieve this. Guideline 5 suggests placing process topics later in the curriculum, when students may have enough maturity and background to appreciate them, but that approach only partially addresses the problem.

This challenge reflects an issue that is of broader concern to academics: How to improve the synergy and communication between industry and academia. The number of industrialists attending conferences such as ICSE is lower than it was many years ago, reflecting different interests in the two communities.

Although significant numbers of academics do work on industry-sponsored research, many have trouble 'penetrating' all but a few narrow areas of a company, let alone whole industry sectors. If they were able to do this, they could learn what is really going on and not only contribute their expertise better to industry, but also take up-to-date know-how and case studies back to the classroom. The unfortunate current situation is perhaps largely due to the need for industrial confidentiality as well as the entrenchment of the status quo on the part of both communities.

There are some positive aspects: In addition to doing some industrial research, many institutions and conferences invite speakers from industry; also students immerse themselves in industry through co-op and internship programs. The range of these interactions has become clear during a series of workshops held at CSEE&T 2006, ICSE 2006 and the 2006 IFIP World Computer Congress. However, what has emerged from these events is that there is a lack of documentation on such interactions that could support an analysis of them.

There has been some research in the area, such as the use of a variety of methods and approaches to support effective student learning on issues associated with ethics and professional practice [62, 63]. These include the use of: discussions based on instructors' personal experiences, active reviews and evaluations of the various codes of ethics, case studies, role-play, games, and web-based learning systems.

One suggested approach involves distributing discussion of process and professionalism issues throughout the curriculum, i.e., finding "teachable moments" in many courses to make connections to industrial practice. However, this requires particular teaching skill, and it is also difficult to assess the extent to which it is in fact being performed.

3.3.1. Action items for the community:

- Better document interactions with industry that influence education, and publish details of how the successful interactions work so they can be emulated.
- Launch a large collective effort to establish two-way knowledge and skill exchanges with companies from more industrial sectors, and with more professionals within these companies.

3.3.2. Research questions:

- What industrial practices are currently not being taught? How effective are these practices? Which of them should be taught to undergraduates?
- To what extent should the hard-to-teach process topics be taught to undergraduates? Everybody acknowledges that the material is important for practice. But perhaps it can be learned reasonably well on the job, so increased emphasis in undergraduate programs may not be necessary, or may not be the best use of educational resources.
- How can process topics be taught better? What types of case studies, simulations or other exercises will work most effectively?

3.4 Defining curriculum standards that are forward-looking

Curriculum standards, like any other standards of a technical nature, are developed with broad community involvement and using a process intended to produce a result that reflects the collective wisdom of the community. The normal result of such a process, which was used in the development of SE2004, is a consensus-based description of the *current* state of typical degree programs.

The most optimistic interpretation of such standards is that they describe current best practice and educational approaches at the time they are developed.

Such standards typically exert considerable influence on degree programs being created or modified for at least a period of five to 10 years after their publication. Students entering such programs require at least another four or five years to graduate.

In an evolving field such as software engineering, the ideal result actually would be curriculum recommendations that are *anticipatory* rather than an

IEEE COMPUTER SOCIETY

affirmation of the present (and even of the past!) The challenge is thus finding a way to design curriculum guidelines for the future while supporting current practice and recognizing that no truly radical curriculum approach can survive the consensus-building process.

One key to finding such a process may begin with acceptance that it is important for developers of curriculum guidelines to really understanding current software development practices (as discussed in the last section), particularly the most effective ones. The primary developers of most curriculum guidelines are typically academics, working with only limited inputs from practitioners. Academics tend to focus on the "knowledge" part of a curriculum, tending to want students to know as much as possible, particularly about their own specialties. Greater consideration of software development practices, however, would force a greater discipline on the process of specifying all of the "stuff" that students need to know. However, this still will not make the curriculum guidelines that result any more forward-looking.

The next necessary step is the development of an understanding of how software development practices have evolved and thus how they relate to earlier approaches. From this understanding comes the potential to predict the evolution of such practices. To make this more concrete, consider how programming practice has changed over the last few decades. We use higher-level languages, increasingly complex APIs, and tools that automate the generation of code in some well-understood domains. The net result is that while programmers may not be more productive in terms of the number of lines of code they can write in a given time period, it is certainly true that much more complex tasks can be implemented by writing a given amount of code. It seems unlikely that this trend will suddenly stop. In fact, one can argue that it will accelerate [64]. Therefore forward-looking curriculum recommendations must prepare graduates who not only can work in the current world of software development, but also possess the skills to handle tools that are more specification-driven. Such graduates should expect that their careers will require them to master an evolving toolset.

3.4.1. Research questions:

- How can we achieve the level of understanding of industry practice necessary to create forward-looking curricula, taking into account the constraints discussed above?
- What curriculum innovations should be considered? These need to be piloted and evaluated before it would be reasonable to embed them in a standard.

3.5 Educating the existing practitioners

In 2004 a working group composed of members of the Royal Academy of Engineering and the British Computer Society produced a report entitled "The Challenges of Complex IT Projects" [65]. The report was based on evidence collected from more than 70 software engineers, managers and academics.

The report highlights, "that education, rather than technology, holds the key to improvements in software project success rates," and "Education is required at all levels, from senior directors to end users." The report also states, as its prime conclusion: "The levels of professionalism observed in software engineering are generally lower than those in other branches of engineering, although there are exceptions." Since computing is a global discipline, there are no reasons to believe that these findings are particular to the UK. It is thus clear that there is a worldwide imperative to direct significant educational resources towards meeting the needs of existing practitioners. In fact, one could argue that more educational resources should be devoted to supporting existing practitioners than those currently in full-time education.

To effectively support the global software sector there must be an international workforce of computing practitioners whose qualifications and expertise can be recognized from country to country. During the 1990's work on providing an infrastructure that would support such a workforce was undertaken under the auspices of the International Federation for Information Processing (IFIP). This work was encouraged by the World Trade Organization which was promoting the view that, in an era of international free trade treaties, and the free movement of workers from one country to another, the establishment of standards regarding the qualifications of professionals is very important.

IFIP's work resulted in the release of a document [66] in 1999 that attempted to define an international standard for professional practice addressing six specific areas: Ethics of professional practice, established body of knowledge, education and training, professional experience, best practice and proven methodologies, and maintenance of competence. In 2000, a series of activities was initiated to both promote and evaluate the IFIP standards document within the SE community. A summary of the activities and the major findings were presented at IFIP's 2005 World Conference on Computers in Education [67]. The overall reaction was very encouraging. However, the areas that were seen as presenting particular challenges with regard to practitioner abilities were those associated with the IFIP statements that:

- Practitioners should be familiar with current best practice and relevant proven methodologies, and

IEEE COMPUTER SOCIETY

- Practitioners must be able to provide evidence of their maintenance of competence.

The need to address the areas of professionalism, best practice, and competence are constant themes throughout the "Challenges of Complex IT Projects" report [65] referred to above, and in the UK at least there has been a recognition that there must be a significant improvement in the quality of both product and service within the computing sector. The British Computer Society (with over 50,000 members in over 100 countries) is currently addressing professionalism and the competence of practitioners with an ambitious three-year project: "The Professionalism in IT Programme" [68]. The project commenced in 2005; a major milestone occurred in May 2006 with the creation of the "Professional IT Programme Alliance" [69] which will work towards the goal of establishing a true IT profession – a profession in which software engineers must play an essential role.

Certification such as CSDP, and the need for licensing in some jurisdictions, as discussed in Section 2.3, should also have an important influence on encouraging software engineers to improve their education levels.

3.5.1. Action items for the community: What are the challenges for the future?

- To convince practitioners, employers and governments that a professional workforce is essential.
- To provide appropriate education for existing practitioners via accredited Continuous Professional Development (CPD) programs. The development of such CPD programs by universities could be one way of mitigating the continuing decline in undergraduate numbers.
- To ensure that practitioners do maintain their competence, perhaps by a certification or licensing process that requires this.
- To ensure that the industry's best practices are recognized, and via CPD programs, are communicated across the profession.

3.5.2. Research questions:

- What are the most appropriate and effective mechanisms to deliver education to existing practitioners in differing situations?
- What practices need to be communicated to a workforce in a constantly changing technical environment? Questions that relate to this are: How to test for best practice? How should best practice be recorded? What are the relationships between best practices, standards, and Bodies of Knowledge – how does each influence the other? What are the roles for professional and international bodies with regard to best practices? Who should be the "keeper" of best practice recommendations?
- In our education programs, how should we prepare our future practitioners for the life-long learning that is essential to support their future careers?

3.6 Making SE education evidence-based

Everything in the design and delivery of software engineering education must become increasingly based on carefully-evaluated evidence.

As discussed in the previous three sections, it is critical to pay more attention to what actually works, or doesn't work, in industrial practice. Doing so will require gathering evidence from broad industry groups, as was done in the previously cited Royal Academy of Engineering study [65]. The accumulation and evaluation of such evidence should be systematic, rather than informal, as is most common for industry/academia interactions about curriculum needs. The result should be a more realistic understanding about how future graduates should be educated to meet the needs of the workplace.

A recent initiative of the National Science Foundation, the CISE Pathways to Revitalized Undergraduate Computing Education (CPATH) program [70], presents an explicit challenge to the academic community in the U.S. to examine computing education models with workforce needs in mind: The CPATH vision is a U.S. workforce with the computing competencies and skills imperative to the nation's health, security and prosperity in the 21st century. This workforce includes a cadre of computing professionals prepared to contribute to sustained U.S. leadership in computing in a wide range of application domains and career fields, and a broader professional workforce with knowledge and understanding of computing concepts, methodologies and techniques.

This multi-year program will provide the computing education community with resources to explore and evaluate new educational approaches. In the case of software engineering, it could enable research into the evaluation of teaching methods and curriculum models, so that the way we teach can be based on real evidence, rather than academic opinions. The CPATH program puts considerable emphasis on industry involvement to insure the relevance of our educational approaches.

3.6.1. Action items for the community:

- Whenever we see an assertion about a 'good' way of doing something in SE or SE education, we should challenge it. Is the asserter in fact basing his or her assertion on concrete evidence that is grounded in industrial experience?

IEEE COMPUTER SOCIETY

- We should ensure that papers accepted in journals and conferences are evidence-based.
- We should consider establishing a program akin to Medicine's 'Cochrane Collaboration' [71] to promote evidence-based decision-making.

3.6.2 Research questions:

- What are the industrial and educational best practices and how do they evolve? How can we demonstrate that they are, in fact, 'best'? How can we best identify, document and disseminate them? If we learn how to work with best practices in the software engineering domain, we may be able to apply a similar approach to the exploration of best practices in education

3.7 Educating the educators

It is unclear what proportion of those teaching core SE courses actually have a deep background in the field. In many universities, professors from a wide range of CS or CE specialties are often deemed competent by deans to teach SE topics. We have heard of instances where an introductory SE course has been taught one year using a very theoretical approach emphasizing formal methods, and another year taught from a hardware-software systems approach.

Deans and department heads must be aware of the potential for such distortions, and work towards both hiring faculty with backgrounds in core areas of SE, as well as enabling existing faculty to upgrade their SE knowledge.

Of course, something that would make a big difference would be to populate faculty ranks, over time, with people who obtained PhD's specifically in the area of software engineering. As discussed in Section 2.2.4, at the time of writing there were four software engineering PhD programs in the U.S.

3.7.1. Action items for the community:

- Develop more SE Ph.D. programs.
- Recognize the problems caused by requiring non-SE faculty to teach SE courses.
- Hire enough SE faculty to teach the required courses.
- Provide funding for those teaching software engineering to attend educational conferences.
- Encourage those who are new to teaching software engineering to attend the "Academy of Software Engineering Education and Training," held each year in conjunction with CSEE&T.

3.7.2. Research questions:

- Given that it will be many years before most people teaching software engineering courses will have SE degrees, what SE-specific knowledge is most important for teaching such courses?
- What are the most common knowledge "gaps" among those teaching software engineering?
- What techniques are most effective at improving the knowledge base of those teaching SE courses?

3.8 Raising the quality and prestige of educational research

Researching how to better educate is essential to improving education. Unfortunately, educational research in most computing departments is not as highly rated as technical research. This is partly a cultural issue with no easy solution. It is also partly related to the quality of the research and the lack of knowledge of what ought to constitute good educational research.

A significant percentage of educational papers submitted for review tend to have one or more of the following major flaws: background literature is ignored or not cited, hypotheses and methods are not stated adequately, and evaluation of results is inadequate. Reviewers tend to recognize these flaws in a technical paper, yet there is inadequate recognition that these flaws are equally unacceptable in an education paper.

Other professions have an extensive and well-respected record in educational research, and have developed high standards for what constitutes a good paper. Virginia Commonwealth University has compiled a list of 13 dedicated medical education journals and many others that do accept education articles [72]. This proves that educational research can be made respectable. One of the problems is funding: Granting agencies only sporadically support educational research, and industry tends only to be interested in those aspects that have to do with training.

Some progress is being made: In the UK there is a system operated under the auspices of Higher Education Academy that rewards outstanding educators and enables them to undertake research relevant to their professional activities. Recipients of these National Teaching Fellowships receive substantial funding to support their research and personal development. Teams including a Teaching Fellow can bid for projects worth up to £200,000 [73].

3.8.1. Action items for the community:

- Raise the awareness of educational research, both in computing and education departments. When tenure and promotion guidelines are being created, ensure that educational research is explicitly considered a valid form of research.
- Lobby for recognition of educational research when communicating with funding agencies.

IEEE COMPUTER SOCIETY

- Arrange for better synergy between SE educational researchers and researchers in education departments. There is a need for SE educators to learn more about learning theories, which play a critical role in framing and understanding contributions, as well as structuring evaluations. SE researchers and educators do not currently tend to know or employ these theories, but definitely should, not just in evaluation, but also in understanding where we can focus our improvement efforts.
- Arrange for multi-university collaborations to fund research that can then be applied in those universities.
- Increase the quality of the research itself so it becomes more respected. Guidelines have been written for what constitutes a good educational paper [74], however program committees need to disseminate these to authors and reviewers. Shepherding of educational papers might also be useful.

3.8.2 Research questions:

- What are the institutions that give the greatest value to educational research in computing? What are the factors involved so that similar approaches can be promoted at other institutions?
- What are the disciplines with robust educational research communities? In these, how is educational research evaluated rigorously and how is high quality research thus promoted?

4. Conclusions

The majority of quality and budgetary issues with software have their root cause in human error or lack of skill. These in turn arise in large part from inadequate education. Therefore improving education should go a long way, in the long run, towards improving software and software practice. We posit that improving education may be the single best investment our profession can currently make.

Focusing on education can also have direct benefits on practice: Many important ideas in science and technology have been developed and refined by educators, attempting to determine how to explain complex concepts.

What are the obstacles to an increased focus on education? We outlined several interrelated challenges: Firstly we need to attract brighter and larger numbers of students. Then we need to educate them better – focusing on the appropriate industry-relevant topics geared towards future needs. We need to communicate the topics better, and ensure that what we teach has evidence to support it. In addition to educating new students, we need to raise the knowledge level of the existing workforce and of the educators themselves.

Tackling all of the above needs research; however, the quality and respect of educational research is a challenge. All SE researchers need to consider doing a certain amount of educational research and to encourage their peers to do so as well.

Acknowledgements

We thank André van der Hoek for his input.

References

[1] Problem Based Learning (PBLK) Home Page – Queen's University, Faculty of Medicine, http://meds.queensu.ca/medicine/pbl/pblhome.htm

[2] K. Leung, K. Lu, T. Huang, B. Hsieh, "Anatomy Instruction in Medical Schools: Connecting the Past and the Future", *Advances in Health Sciences Education*, 11, 2, May 2006, Springer, 209-215

[3] F. Brooks, *The Mythical Man-Month*. Addison Wesley, 1995.

[4] P. Bourque and R. Dupuis, eds, IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge: 2004 Version*, www.swebok.org, April 2005.

[5] IEEE/ACM Joint Task Force on Computing Curricula. *Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, IEEE Computer Society Press and ACM Press, August 2004., http://sites.computer.org/ccse/

[6] "Certified Software Development Professional." *IEEE Computer Society*, Los Vaqueros, CA. http://computer.org/certification/

[7] J. Sargent, "An Overview of Past and Projected Employment Changes in the Professional IT Occupations" *Computing Research News*, 16, 3, May 2004, pp. 1, 21.

[8] J.E. Tomayko, "Forging a discipline: An outline history of software engineering education", *Annals of Software Engineering,* 6, 1998, Springer, pp. 3-18.

[9] I. Sommerville, *Software Engineering, 8th Edition*, Pearson, 2006, www.pearsoned.co.uk/sommerville

[10] R. Pressman, *Software Engineering: A Practitioner's Approach*. 6th Edition, McGraw Hill, 2004.

[11] J.B. Thompson, "A Long and Winding Road (Progress on the Road to a Software Engineering Profession)," *COMPSAC 2001*, Chicago, IL, Oct. 2001, pp. 39-45

IEEE COMPUTER SOCIETY

[12] D. Gotternbarn, "How the New Software Engineering Code of Ethics Affects you." *IEEE Software*, Nov/Dec 1999, pp 58-64.

[13] R. Davis "PEs Explore Credentialing of Software Professionals." *Engineering Times*, May 2001, p. 10.

[14] G. Engel, "Program Criteria for Software Engineering Accreditation Programs." *IEEE Software*, Nov/Dec 1999, pp 31-34

[15] T.C. Lethbridge, "What Knowledge is Important to a Software Engineer?", *IEEE Computer*, May 2000, pp. 44-50.

[16] T.C. Lethbridge, "Priorities for the Education and Training of Software Engineers", *Journal of Systems and Software*, 53,1, 2000, pp. 53-71.

[17] P. Bourque, R. Dupuis, A. Abran, J. Moore, and L. Tripp. "The Guide to the Software Engineering Body of Knowledge." *IEEE Software*, Nov/Dec 1999, pp 35-44.

[18] ISO, *Software Engineering - Guide to the Software Engineering Body of Knowledge (SWEBOK),* ISO/IEC 19759:2005.

[19] IEEE/ACM Joint Task Force on Computing Curricula., *Computing Curricula 2005, the Overview Report*, http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf

[20] T.C. Lethbridge, R. LeBlanc, A. Sobel, T. Hilburn, and J. Díaz-Herrera, J., "SE 2004: Recommendations for Undergraduate Software Engineering Curricula", *IEEE Software*, Nov-Dec 2006, pp. 19-25.

[21]J.M Atlee, R.J. LeBlanc, T.C. Lethbridge, A. Sobel, and J.B. Thompson, "Reflections on Software Engineering 2004, the ACM/IEEE-CS Guidelines for Undergraduate Programs in Software Engineering", in *Software Engineering Education in the Modern Age, Lecture Notes in Computer Science, Volume 4309/2006*, Springer, 2006, 11-27.

[22] T. Dyb, M. Jrgensen and D. Sjøberg, "Empirical methods in software engineering research", *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007

[23] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams", *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007

[24] R.N. Taylor and A. van der Hoek, "Software Design and Architecture: The once and future focus of software engineering",Future of Software Engineering 2007, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007

[25] R. Shackelford, ACM Education Board Annual Report. FY 2004, www.acm.org/about_acm/commreports/fiscal_year_2004/ed_board_FY2004.pdf

[26] P. Freeman, A.I. Wasserman, R.E. Fairley, "Essential Elements of Software Engineering Education", *ICSE 1976*, pp. 116-122.

[27] P. Freeman, A.I. Wasserman, "A Proposed Curriculum for Software Engineering Education", ICSE 1978, pp. 56-62

[28] M. Shaw, "Software Engineering Education: A Roadmap". *ICSE 2000, Future of SE*, 2000, pp. 371-380

[29] "Department of Software Engineering." Rochester Institute of Technology, Golisano College of Computing and Information Sciences. Rochester, NY. http://www.se.rit.edu/

[30] "Software Engineering at Monmouth." Monmouth University, Long Branch, N.J. http://www.monmouth.edu/se/

[31] "Software Engineering." Milwaukee School of Engineering. http://www.msoe.edu/eecs/se/

[32] "School of Computing and Software Engineering." Southern Polytechnic State University, Marietta, GA. http://cs.spsu.edu/csdept/

[33] D.J. Bagert, S.V. Chenoweth, "Future Growth of Software Engineering Baccalaureate Programs in the United States", *2005 ASEE Annual Conference & Exposition*, Portland, OR; USA; June 2005. 8 pp

[34] Canadian Engineering Accreditation Board, Accredited Engineering Programs, http://www.ccpe.ca/e/acc_programs_2.cfm

[35] CIPS, Accredited Programs (Software Engineering), http://www.cips.ca/standards/accreditation/csac/default.asp?load=SE_accredited

[36] A. Finkelstein, "European Computing Curricula: A Guide and Comparative Analysis", *Computer Journal*, 36, 4, pp 299-319, 1993.

[37] A. Cowling, "The First Decade of an Undergraduate Degree Programme in Software Engineering", *Annals of Software Engineering*, 6, pp 61-90, 1998

[38] BCS and IEE, *A report on Undergraduate Curricula for Software Engineering*, The British Computer Society and The Institution of Electrical Engineers, June 1989.

[39] J.B. Thompson and H.M. Edwards, "Software Engineering in the UK 2001", *Forum for Advancing Software Engineering Education (FASE)*, 11, 119 (141st issue), Nov. 15th 2001, pp 1-18.

[40] Quality Assurance Agency for Higher Education, www.qaa.ac.uk

[41] "Software Engineering Programs." Naval Postgraduate School, Monterey, CA. http://seac.nps.navy.mil/

[42] "Doctoral Programs, Software Engineering." Carnegie Mellon University. http://www-2.cs.cmu.edu/afs/cs.cmu.edu/Web/education/doctoral_isri.html/

[43] BLS, Occupational Outlook Handbook, 2006-07 Edition. US department of Commerce. "Tomorrow's Jobs."

[44] BCS, "The Professionalism in IT Programme", *IT Now*, May 2006.

[45] IET.tv (website)

[46] The Online Ethics Center for Engineering and Science, Case Western Reserve University, www.onlineethics.org.

[47] The Network Community for Software Engineering Education, www.swenet.org

[48]S.B. Seidman. And J.F. Naveda, *IEEE Computer Society Real-World Software Engineering Problems: A Self-Study Guide for Today's Software Professional,* Wiley, 2006.

[49] Conference on Software Education and Training, http://conferences.computer.org/cseet/

[50] UK Computing Research Committee, Grand Challenges in Computing Education. http://www.ukcrc.org.uk/grand_challenges/index.cfm

[51]National Science Foundation. "Report of the NSF Workshop on Integrative Computing Educational and Research Northwest Workshop." Boston. November 2005.

[52] Money Magazine, "50 Best Jobs in America", http://money.cnn.com/magazines/moneymag/moneymag_archive/2006/05/01/8375749/index.htm

[53] J. Foley. "Computing, we have a problem", May 2005 *Computing Research News*, May 2005, vol. 17, No. 3.

[54] W. Asprey, F. Mayadas, and M. Y. Vardi (Eds). "Globalization and Offshoring of Software." ACM, Job Migration Task Force. 2006, http://www.acm.org/globalizationreport/.[

[55] P. Krill, "Survey: Offshoring does not cost developer jobs", InfoWorld, http://www.infoworld.com/article/07/01/11/HNsiaa_1.html, Jan 11, 2007.

[56] "Science 2020: " Microsoft Research,.Cambridge, UK. 2006, http://research.microsoft.com/towards2020science/

[57] F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer*, Vol. 20, No. 4 (April 1987) pp. 10-19.

[58] Indiana School of Informatics. http://www.informatics.indiana.edu/

[59]Cornell Faculty of Computing and Information Science. http://www.cis.cornell.edu/

[60]BS in SE, Applications Domains. RIT http://www.se.rit.edu/appdomain.php

[61] D. Carr and R.J. Kizior, "The case for continued Cobol Education", *IEEE Software*, 17, 2, Mar-Apr 2000, pp. 33-36.

[62] W. Towell, "Teaching Ethics in the Software Engineering Curriculum", *16th Conference on Software Engineering Education & Training*, 2003, Madrid, Spain, pp 150- 157.

[63] E. Towell, and J.B. Thompson "A Further Exploration of Teaching Ethics in the Software Engineering Curriculum", *17th Conference on Software Engineering Education & Training*, 2004, pp. 39-44.

[64] D.S. Batory, "Feature-Oriented Programming and the AHEAD Tool Suite". *ICSE 2004*, 702-703

[65] RAE and BCS, *The Challenges of Complex IT Projects, A report by members of the Royal Academy of Engineering and the British Computer Society*, The Royal Academy of Engineering, 2004, http://www.bcs.org/upload/pdf/complexity.pdf

[66] I. Mitchell, P. Juliff, and J. Turner, "Harmonization of Professional Standards", International Federation of Information Processing, 1998, http://www.cet.sunderland.ac.uk/seis/icse2001workshop/IFIPharmonisationDraft1998.htmlAlso, available as an appendix to [67]

[67] J.B. Thompson, "Evaluations of IFIP's Proposed Standards for Professionals", *8th IFIP World Conference on Computers in Education, (WCCE 2005)*, University of Stellenbosch, Cape Town, South Africa, July 4-7, 2005

[68] British Computer Society, *The Professionalism in IT Programme*, Swindon, UK, http://www.bcs.org/upload/pdf/summitreport.pdf

[69] British Computer Society, ProfIT 2006 Conference, London May 2006, www.profitalliance.org.uk

[70] CISE Pathways to Revitalized Undergraduate Computing Education (CPATH). Program Solicitation NSF 06-608, September 2006, http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=500025&org=CISE

[71] Then Cochrane Collaboration: the reliable source of evidence in health care. www.cochrane.org.

[72] Journals Publishing Medical Education-Related Articles, http://www.library.vcu.edu/tml/bibs/medicaleducationjournals.html

[73] UK Higher Education Academy, National Teaching Fellowship Scheme, http://www.heacademy.ac.uk

[74] T.C. Lethbridge, and D. Port, "A Brief Guide to Researching and Writing for CSEE&T", http://www.site.uottawa.ca/cseet2005/CSEETResearchGuide.pdf

IEEE COMPUTER SOCIETY