The Push to Make Software Engineering Respectable

A recognized engineering profession must have an established body of knowledge and skill that its practitioners understand and use consistently. After 30 years, there is still a wide gap between the best and the typical software engineering practices. To close this gap, we need a deeper partnership among industry, academia, and professional societies.

Gilda Pour
San Jose State University

Martin L. Griss
Hewlett-Packard Laboratories

Michael Lutz
Rochester Institute of Technology

Software engineering (SE) is maturing as a discipline and profession, but three decades after the first NATO Conference on Software Engineering, it is still not regarded as a legitimate, respectable engineering profession. In 1995, Gary Ford and Norman Gibbs of the Software Engineering Institute evaluated what it means for a profession to be mature and how SE was doing.[1] Their extensive and fascinating study found that, relative to other fields and engineering branches, most elements that make SE a profession were quite immature.

Five years later, SE has countless practitioners (a.k.a. software developers), thousands of articles, dozens of conferences and workshops, and a respectable number of education and training programs. The computer science (CS) and SE communities (educators, researchers, practitioners, and professional societies) are further along in defining a body of knowledge, code of ethics, accreditation guidelines, and licensing programs.

But despite all this progress, SE, while recognizable, is still immature—as evidenced by the significant gap between vision, education, and standard practice. The reasons are legion, but they boil down to one simple fact: The field is still young. There just hasn't been enough time to gain widespread community consensus on issues, to stabilize a core body of knowledge, and to develop a large enough pool (several generations) of experienced practitioners and educators. The rapid changes in this field make the road to maturity even rougher.

Although we believe time will eventually mature SE, a calculated push can accelerate the maturation process. By "push," we mean defining, accrediting, and evaluating new curricula that stress CS and SE fundamentals and practice, focus on lifelong learning and team experience, and increase the role of the professional societies in accreditation, certification, and licensing efforts. Although the push will not overcome all issues, it will go far in addressing the most knotty ones.

Establishing SE programs is fraught with economic, political, and pedagogic challenges—notably how to divorce SE (or *if* it should be divorced) from existing CS and computer engineering (CE) curricula. Any solutions will have to come from deeper, more extensive industrial-academic-professional society partnerships. This is key.

We have spent some time considering the reasons for SE's immaturity. All of us are heavily involved in both industry and academia and have been active in professional societies that aim to promote SE as a profession. Promotion efforts are by no means limited to the US, but because our experience is primarily with US activities, that is our focus in this article. Our main goal is to explore, from a multifaceted perspective, *why* we are where we are now and *how* we can move forward.

WHAT MAKES A PROFESSION MATURE?

As Table 1 shows, a mature profession must have several key infrastructure components. In addition to solid education programs, proper guidance from industry and professional societies is critical to adequately prepare graduates for entrance to an SE profession. Accreditation of programs and possible certification and licensing of graduates safeguard the quality of any education and training program and provide assurance that graduates meet and maintain requisite levels of knowledge and practice.

The various infrastructure components work together to ensure that those entering the profession become familiar with the currently accepted body of knowledge and that they practice it in a manner consistent with the tenets of the profession. Research and practice will evolve the body of knowledge; thus, practitioners must continue to enhance their skills and practice as they gain enough experience to specialize. Table 1 indicates the relative maturity of each infrastructure component, using Ford and Gibbs' four-level scale.[1] However, the ratings do not reflect the consensus or breadth of compliance on a particular component, which tends to be low.

A major symptom of a not-quite-mature field—and a frustrating one to address—is the wide gap between education and practice and between best and typical practice. Ongoing efforts in many maturity elements are attempting to narrow these gaps.

PROFESSIONAL SOCIETIES

There are many engineering and CS professional societies, but the two most clearly identified with the SE profession are the Association for Computing Machinery (ACM) and the IEEE Computer Society (IEEE CS). Recently, the two joined forces to define SE as a profession, provide a code of conduct for professionals, and formulate appropriate criteria for accrediting SE educational programs. The Software Engineering Coordinating Committee (SWECC) defines the scope of these constituent tasks and monitors and coordinates the working groups. SWECC membership is divided evenly between the ACM and the IEEE CS—a balance that is the hallmark of SWECC-commissioned activities. Table 2 lists some of SWECC's current projects.

Although the ACM and the IEEE CS sponsor SWECC projects, international participation has been uncommonly strong. More important, professional groups in other countries are beginning to take SE seriously: Graduates of UK computing programs that are accredited by the British Computer Society are eligible for Chartered Engineer status, and licensing for graduates of accredited SE programs in Australia is similar to licensing for other engineering graduates. Canada also seems to be leaning toward similar treatment of SE as a profession.

Barriers. Activities to develop a body of knowledge, accreditation, curricula, a code of ethics, and licensing and certification are ongoing but are proceeding rather slowly, staffed primarily by volunteers. Although the IEEE CS and the ACM have actively promoted professionalism in SE since 1993, they do not agree on all issues related to SE as a profession. Their positions on licensing professionals and the pace of accreditation are far apart, for example.

Table 1. How does software engineering rate on the maturity scale?

Infrastructure component of a mature profession	How software engineering measures up*
Recognized body of knowledge	IEEE-CS/ACM task force has released a first body of knowledge report with consensus expected to take at least 4 years (level 2-3)
Professional society/societies	IEEE-CS, ACM, and SIGSOFT, active, a specific SE society under discussion (level 2-3)
Code of ethics	Recently developed for SE, but not widely known or practiced (level 1-2)
Initial professional education system	Some SE BS degrees recently offered by CS departments and special SE departments in the US, the UK, Europe, Canada, and Australia (level 1-2)
Accreditation of professional education programs to improve program quality and ensure some uniformity	Accreditation Board for Engineering and Technology (ABET) criteria for SE are in place, which the Computer Science Accreditation Board (CSAB) will evolve (level 1-2)
Skills development mechanism for professionals entering the practice	Several MSE programs, certificate short courses, and so on (level 1)
Professional development programs to maintain currency of knowledge and skills	Fragmented offerings—extension courses, seminars, professional conferences, manufacturer certification programs (level 1)
Certification of professionals administered by the profession	Limited, inconsistent, technology-based; some certificates issued by manufacturers (for example, Microsoft MCSE) (level 0-1)
Licensing of professionals administered by government authorities	Recently in the US (Texas), Canada, and the UK (level 1-2)

*Level 0 = Nonexistent.
Level 1 = Ad hoc, some form of element exists, but is not identified with the SE profession.
Level 2 = Specific, the element exists and is clearly identified with the SE profession.
Level 3 = Maturing, the element has existed for many years, is continually improving, and is under active stewardship of an appropriate professional body.

Table 2. Software engineering projects jointly sponsored by the ACM and the IEEE CS through the Software Engineering Coordinating Committee.

SWECC project*	Description and status as of April 2000
Software Engineering Body of Knowledge (SWEBOK)	Create an index to the core body of knowledge (BOK), structure it, refine with specialist area committees, and gain community consensus. — *Phase two BOK draft (Stoneman) essentially complete.*
Software Engineering Code of Ethics and Professional Practice (SWCEPP)	Create an expanded and detailed code of SE ethics for educators and practitioners based on a shorter engineering code of ethics. Provide case studies and training materials to help rapidly educate community. — *Final draft submitted for approval.*
Software Engineering Education Project (SWEEP)	Define model accreditation criteria and sample curricula consistent with SWEBOK for BS and other SE education programs. — *SWECC approved accreditation guidelines in December 1998.*

* More details and current status of key activities are described in the *IEEE Software* special issue on professional software engineering (Nov./Dec. 1999) and at http://www.computer.org/tab/swecc.

At the national level, the President's Information Technology Advisory Council (PITAC) report articulated the urgency of SE issues, and Congress approved money for the National Science Foundation (NSF), but these recommendations are not coordinated with the societies, nor do final proposals really target funds toward SE issues. Thus, progress is limited to the rate at which society members and leadership can change.

Possible solutions. The societies, via member involvement, must strive to create a broader consensus on the core of the SE profession. Perhaps the SE community needs to address issues more aggressively. We could accelerate progress with full-time paid staff, for example, rather than relying on the efforts of part-time volunteers.

Body of knowledge

In long-established professions, such as medicine, law, or civil engineering, the body of knowledge was codified as part of a long maturation process. To accelerate the maturation of new professions, professional communities can consciously and explicitly define the body of knowledge, rather than waiting for a natural evolution. The codification proclaims what is unique about the profession, demarcates the boundaries with related professions, and significantly aids education, certification, and licensing. Getting practitioners and educators to agree on the core body of knowledge is a key milestone in any discipline.

SWECC established the Software Engineering Body of Knowledge (SWEBOK) project to collect and document the state of SE practice, using a three-phase consensus-building approach.[2] So far, the SWEBOK project has identified and is structuring many topics that texts and SE programs cover. Area committees are expanding and distilling the material, using public reviews and surveys to reach consensus. The goal is to categorize the material as core, advanced, or research to determine what should be taught and known at various professional development levels.

It will take at least four years to reach initial consensus, with ongoing refinement and evolution. The first two phases—strawman and stoneman—are essentially complete, providing a basis for significant work on model curricula and certification.

Barriers. Achieving consensus on the common core takes time, as will agreeing on related specialties. More time will elapse before curricula incorporate this core, especially when state education processes are involved.

Possible solutions. Some ongoing activities are increasing awareness and buy-in. Adding workshops and panels at major conferences could heighten awareness even more. Support for innovative programs can accelerate the rate of change. Also, significant industry, government, and society sponsorship and funding can help create sponsored and widely advertised pilot or magnet programs to help jump-start change, as well as offer fast-track certification programs. Neither academia nor industry will change overnight, however. Current mind-sets must change, which may require new blood in upper ranks.

Code of ethics and professional practice

SWECC established the Software Engineering Code of Ethics and Professional Practice (SWCEPP) project to develop a code that the SE community as a whole would find acceptable. An international effort produced a detailed add-on to the standard engineering code of ethics, which differs across countries.[3] Some decry the extra detail as unnecessary, claiming that we already have an adequate engineering code. We agree with others who believe the extra detail and clarity *are* needed, both to enhance education and to clarify to SE educators and practitioners that they are indeed engaged in an engineering effort that affects society.

The ACM and the IEEE CS recently approved the draft code, with the goal of having it recognized as appropriate for all involved in SE. There are as yet no formal consequences for professionals who violate the code. In other professions, a licensing board hears accusations and can recommend disbarment (as in the legal profession) or some other sanction. The code could also be used in legal proceedings to determine whether or not a software engineer has acted in accordance with professional norms and the accepted body of knowledge.

Barriers. Awareness is the biggest problem. Many of those involved in SE still have not heard of the draft code or SWCEPP and are confused about how it affects them. Only a handful of schools yet teach SE ethics and professional practice.

Possible solutions. We need to have more articles and studies at key conferences, active industrial lobbying, wider-spread accreditation. Most of all, we need to allow time for awareness and practice to grow. We can then begin self-monitoring.

Education and accreditation

Many fledgling undergraduate SE programs exist in the US and abroad,[2,4] and we expect to see more in the next few years. The sidebar "Elements of a Good Software Engineering Program" describes what constitutes an effective program. Common to all programs is the recognition that SE is fundamentally different from CS and CE. The sidebar "The Case for Software Engineering Independence" describes some of these differences.

The Software Engineering Education Project (SWEEP)[4] provides a detailed set of guidelines for SE programs that will eventually seek accreditation. SWECC officially adopted the SWEEP accreditation guidelines in December 1998. Also in 1998, the Computer Science Accreditation Board started a formal integration of its operations and criteria with the Accreditation Board for Engineering and Technology—

Elements of a Good Software Engineering Program

A comprehensive undergraduate SE program builds on a traditional CS program, incorporating various components adapted from typical engineering education programs. It has eight main elements:

- *CS fundamentals* provide the core technical knowledge and skills about software and hardware artifacts and techniques to address key technical problems. These include programming languages, modeling, formalisms, mechanisms, databases, operating systems, networking, algorithms, programming, and distributed systems.
- *SE fundamentals* provide the core technical knowledge and process skills and tools to create, manage, and maintain software and documentation and deal with complexity and human error. These include software process discipline, life-cycle models, software metrics and economics, architecture, design methods and skills, design inspections, testing, configuration management, and standards.
- *Engineering practice and ethics* provide an understanding of general systems principles, economic and functional trade-offs, the implication of building artifacts for use, and how engineers serve society and balance their responsibilities to their employers, their customers, and society.
- *Effective communication and teamwork skills* provide knowledge and skills in working with diverse human beings—from peers to management and customers.
- *Experience in an application domain* that exposes students to real-world problems. This element is key to grounding and consolidating core knowledge and skills in a particular area.
- *A significant team project* exposes students to issues such as requirements change, project management, configuration management, use of tools, and team dynamics. The project is typically run under somewhat controlled conditions in a laboratory or studio setting and can be completed in assigned time, with mentors oriented to the educational experience.
- *Experience in some industrial setting*, perhaps through a required internship or co-op program, provides even more exposure to real-world issues, people, pragmatics, and the vagaries of real projects under real-time pressure.
- *Tools for effective lifetime learning*, including experiences that rehearse how to seek, evaluate, and use information not directly provided by assigned texts and lectures. For example, projects might require students to find information on the Web or in the library, evaluate and integrate possibly conflicting materials, and attend and report on a conference and a tutorial. Students might also participate in an ongoing "journal club" or "research seminar," where they would read, analyze, and compare many papers.

Because software technologies crop up quickly and because IT's role is constantly changing, any SE program *must* emphasize lifelong learning. Numerous phenomena and issues (open source, the Web, component- and service-oriented computing) will affect SE, and SE professionals must be able to understand and evaluate those effects.

Also, different parts of software construction require a different mix of skills: For some segments, a laboratory-style team project is important (do an experiment, measure it, evaluate it); in others a studio approach is better (do creative work in some medium under the watchful eye of a mentor or instructor).

a merger that will unify the criteria and process used to accredit SE programs. SWEEP will use the accreditation guidelines as a specification to design one or more model curricula for SE, leveraging the initial work of SWEBOK and the Computer Curriculum 2001 task force (recently formed by the IEEE CS and the ACM to review and upgrade the 1991 computing curricula), among others.

In other countries, the development of undergraduate SE programs is even further along. Australia and the UK, for example, established initial programs in the early 1990s. As a result, both countries have accreditation criteria for SE programs, and graduates have equal professional standing with those in more traditional engineering disciplines. India's software industry is also growing rapidly, in large part because of the disciplined engineering approaches used in Indian firms.

Barriers. Establishing SE in undergraduate (or even graduate) curricula is rarely straightforward. Several educational institutions, including the Rochester Institute of Technology (RIT), have successfully established an undergraduate SE program, but one model is unlikely to work for all institutions. The California state university system, for example, must ensure that appropriate two-year programs are in place in parallel with introducing the four-year degree.[5]

A critical problem is finding faculty interested in and capable of teaching SE because few CS faculty have enough real-world SE experience. Teaching SE

The Case for Software Engineering Independence

Many institutions tend to think of software engineering (SE) as just a kind of computer science (CS) or computer engineering (CE). This leads to problems because the disciplines differ in both focus and approach: SE studies software; CS and CE study primarily hardware, algorithms, and languages. CS and CE develop knowledge; SE applies that knowledge to engineer high-quality software systems. The IEEE Standard 610.12 definition of software engineering states:

> "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software, and (2) The study of approaches as in (1)."

focuses on acquiring and applying technical standards, but does not address *ethical* standards.

Learning and building

The ACM/IEEE CS Task Force on the Core of CS for Computing defines the computing discipline as "the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application." Thus, the scientific question underlying all of computing is "What can be (efficiently) automated?"[1]

Software engineers and computer scientists have fundamentally different goals. As David Parnas says, "Scientists learn science plus the scientific methods needed to extend it," and "Engineers learn science plus the methods needed to apply it."[2]

Debates about the relationship between SE and CS date back to the late 1970s, when Anthony Wasserman and Peter Freeman[3] identified five key areas that provide the foundation for SE: computer science, management, communication skills, problem solving, and design methodology. Mary Shaw,[4] Bill Wulf,[5] and Parnas[2] highlight the intimate connections between (software) science and engineering, and discuss the costs and benefits of separating computing into distinct science and engineering disciplines. Despite the political and emotional reasons to claim that SE should be just a part of CS and CE, the strong differences in the goals and style of education—and the need for professional software engineers—motivate distinct SE programs.

Art and science

There is some controversy about the kind of engineering discipline SE actually is and how it differs from CS. Distinct factions in the software community believe that creating software is primarily an artistic endeavor; others consider it more mathematical, while others believe that process and method are key. But both art and science are at the core of engineering, as attested to by this quote from Henry Petroski, a noted civil and environmental engineer and author of the acclaimed "To Engineer is Human":[6]

> The conception of a new structure can involve as much a leap of the imagination and as much synthesis of experience and knowledge as any artist is required to bring to his/her canvas or paper. Once the design is completed, it must be analyzed by the engineer as scientist in as rigorous an application of the scientific method as any scientist must make.

References

1. P.J. Denning et al., "Computing as a Discipline," *Comm. ACM*, Vol. 32, **No. ?? (or month)**, 1989, pp. **XX-XX.**
2. D.L. Parnas, "Software Engineering Programs Are Not Computer Science Programs," *IEEE Software*, Nov./Dec. 1999, pp. 19-30.
3. A. Wasserman and P. Freeman, "Software Engineering Concepts and Computer Science, Curricula," *Computer,* June 1997, p. 91.
4. M. Shaw, "Prospects for an Engineering Discipline of Software," *IEEE Software*, Nov. 1990, pp. 15-24.
5. W.A. Wulf, "Are We Scientists or Engineers?" *ACM Comp. Surveys*, Mar. 1995, pp. 55-57.
6. H. Petroski, *To Engineer Is Human: The Role of Failure in Successful Design*, Vintage Books, N.Y., 1992, p. 40.

A challenge to proponents of accredited SE education programs is proving that their graduates produce better systems more economically.

requires competence in SE life-cycle processes and so on—things that do not "feel like" CS. Compounding the problem is the tradition of basing faculty hiring on the applicants' research. Typical systems- or theory-oriented colleagues see SE research and practice as fuzzy, decrying the lack of evidence that SE principles and techniques actually work. SE research that could validate efficacy, such as metrics, management, teams, processes, or methods typically involve social-science-like experiments, which bothers many traditional CS researchers. As a result, new or prospective SE faculty often face skeptical or even hostile colleagues.

Another sensitive issue is the independence of the SE program from CS, CE, or any other department.[6] Many institutions resist SE independence, remembering the battle over the EE-CS split. The concern is that SE is an attractive combination of engineering and CS, and that traditional programs will lose resources and students as a consequence.

Possible solutions. Sometimes a partnership with CE, industrial engineering, management, or business can provide the ideal SE instructional team. An interdisciplinary program that incorporates the strengths of both CS and CE can result in a more stable and harmonious environment for both students and faculty.[7] Industry and NSF advocacy and funding of SE programs, projects, or chairs can help increase interest and respect. If industry demanded a more standard, comprehensive, and accredited SE education program (as it does for other engineering professions) and was willing to invest in developing such a program (not just an SE training program to address a programmer shortage), more institutions would comply.

In several regions, local industry does work closely with local educational institutions to help motivate and drive change, but for the most part, industry support and encouragement is still lacking. In all US cases of successful industry support, the program resulted from a small, motivated contingent of faculty who established credibility with the upper administration. Examples are RIT's co-op program,[7] the University of Utah, Georgia Tech, and the University of California at Santa Cruz.

A challenge to proponents of these programs, as well as to researchers, will be to prove that students of these programs produce better systems more economically, relative to untrained students. Anecdotal evidence from the RIT co-op program is encouraging, but it is only a small step in the right direction. True validation of this hypothesis will require cooperative work between academia and industry to gather and analyze data.

Skills development

SE has many facets, and different training will be required for different software roles and specialties, including, but not limited to, architect, system engineer, design engineer, test engineer, quality engineer, maintenance engineer, programmer, and technician. Different problem domains will surely require different specialist skills; consider the difference in content and scale between architecting and developing the user interface (UI) subsystem for a large-scale, multiuser computer game versus a UI system for a reactor or airplane controller. Even more different are the skills needed for UI design versus those for database or operating system development.[8] Programmers must know specific languages and associated tools, and be familiar with the skills needed to apply coding guidelines and standards. A senior software engineer must have both broad technical knowledge of CS and SE principles and be able to apply technical and managerial practices that cover everything from project feasibility to product delivery and ongoing support.

Just as chemical and electrical engineering are treated as distinct fields within "physical engineering," so we could distinguish, educate, and certify well-understood, distinct specialties in SE, such as compilers, databases, and operating systems.[8]

Barriers. SE covers a vast range of subdisciplines, and some senior members of the field propose that "SE for _____" is how we should move forward. However, there is yet no consensus on the core areas, nor on which parts of the core apply to which subdisciplines. Some core guidelines, such as design and code inspections, are probably good for all parts of SE, but some faculty are adamant that good practices like code inspections do not belong in CS. This makes it hard to find a place for these practices if the institution does not endorse a separate degree for SE.

Possible solutions. SWEBOK and SWEEP will help distinguish core, advanced, and special areas and define which curricula contain which parts. This incremental strategy will help support an initial consensus and then broaden that consensus as the experience base widens. We must agree on the body of knowledge and drive model curricula that incorporate core elements and meet accreditation guidelines. We need more effort than current part-time volunteers can provide to make this happen. Greater industry involvement will garner interest, help shape programs, and provide skilled practitioners, along with opportunities to study real-world problems. Fortunately, the SWEBOK project has significant industrial sponsorship.

Lifelong, self-directed education is also important. In a world of free agents and contractors, software engineers must pick up—on their own—many of their specialty skills. Commercial certification (MCSE and Cisco CCIE or CCNA, for example) is valuable, and classes from university extension or private institutions are key to keeping skills current and marketable.[9]

Licensing and certification

Licensing and certification is a significant (but not essential) aspect of an engineer's professional stature. Society increasingly depends on software for a wide variety of mission- and life-critical systems. Concerns are escalating about liability and contracts that call for a certified level of expertise in the software professional. The furor over Y2K has certainly raised awareness of the potential impact of SE design decisions.

The degree of licensing in practice varies from profession to profession. Licensing seems to pertain mostly to those who must sign off on a contracted deliverable or who do bonded private consulting. Most engineers who work for companies are not legally required to be licensed, though many civil and electrical engineers will do so to help advance their careers.

Accreditation, licensing, and certification mechanisms together aim to protect the public's health, safety, and welfare by providing some assurance that a practitioner is competent in the certified or licensed specialty. Licensing also protects members of the profession, both by limiting the number of professionals so licensed and by establishing norms that protect individuals and groups in some liability suits. The legislative bodies of all 50 states and the US territories have created statutes that require an engineer performing work for the general public to be licensed by the state or territory in which the work is being performed. The laws require licensing applicants to meet certain standards of education and work experience and to pass a series of examinations.

Barriers. The SE community has mixed opinions about whether professional licensing at this time is a good idea.[10,11] Does it make sense to license software engineers before making an SE body of knowledge available? The ACM recommended against licensing on the basis of advice from a blue-ribbon ACM committee. This is largely a political issue—people are afraid of being controlled, of limiting access to a scarce labor pool, and of legislating best practices.

Many feel that licensing is premature given the SWEBOK's current state and the absence of corresponding accredited education programs. They doubt what SE practice can actually guarantee. Their concern is that best current SE practices do not result in systems with the same reliability and safety that other engineering disciplines produce.

Possible solutions. The Texas State Board of Engineering Licensing uses an equivalency process to award a professional SE license without examination.[12] Following its law and tradition, Texas requires licensing only for engineers whose services are publicly available. Still, the ramifications of any licensing have led to serious debates within the computing community, which will take time to resolve. Many believe that other states eventually will follow Texas. But regardless of the outcome, society and lawsuits will dictate that we proceed incrementally, starting in safety-critical industries. Many believe that starting licensing now will at least improve the state of the practice and reduce error. Licensing efforts are also under way outside the US, including in the UK and Canada (British Columbia and Ontario).

Involving industry more in SE teaching and research requires proactivity on both sides.

A SUSTAINED PARTNERSHIP

A common theme in solutions to SE maturity barriers is to involve industry more in SE teaching and research. To do so, we need proactivity on both sides. A deep, sustained partnership will encourage the development of more effective SE education programs and ensure that university research will have more access to and influence on industrial-scale development. We get the best of both worlds: industrial involvement and advice and academia's long-term view of what makes a quality education. This emphasis on the long-term view is what differentiates the partnership in education from a training exercise.

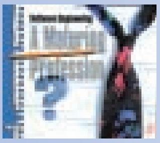
The partnership should focus on helping universities arrive at the appropriate balance between fundamental knowledge and its engineering application. Because many large companies have had to mount significant SE training programs—in large part because of the dearth of SE education in college, they should be more than willing to assist in nurturing SE programs that emphasize developing core skills. Increased collaboration has many immediate benefits, such as reduced training costs for companies and more focused SE research for institutions. Increased collaboration between academia, engineering institutes, and industry would substantially reduce serious mismatches in expectations.

Indeed, effective industry involvement is not trivial. Goals and investments must match—typically, academia wants to train in fundamentals and lifelong learning, while industry focuses on acquiring skills to fill an immediate need.

The sidebar "Building a Strong Industrial-Academic Partnership Now" lists some steps each side can take right away to begin forging this partnership.

Software engineering is an emerging profession that will greatly mature within the next decade. The SE community must define, accredit, and evaluate new curricula, stressing lifelong learning, significant team experience, and practical theory and fundamentals. We must also address the lack of consistency among the undergraduate SE programs that do exist. Educators do not yet agree on the core elements to teach; without systematic accreditation and licensing, there is less pressure to quickly adapt programs to increase consistency and incorporate new knowledge and skills. Academia is slow to incorporate practices that work well (for example, inspections).

Building a Strong Industrial-Academic Partnership Now



Companies and academic institutions can take several immediate steps to align their expectations for the software engineering profession.

If you are in industry:

- Develop relationships with selected universities and SE faculty.
- Provide support for development of educational programs rather than just training programs. This means preparing students for an SE career, not just a job. You could offer guest lectures and mentoring, for example, or participate on an SE advisory board.
- Provide input to the schools on how their graduates fare in industry.
- Create more exciting and educationally aligned internship programs.
- Provide financial support. This *is* important, not only because of its direct value, but also because it is a measure of respect. Even if you can't fund a big project, you can fund a small collaborative research project and invite the faculty and students to work with you. Be sure to provide access to real data and project records. Sanitize the data and records to remove confidential information as needed or work under nondisclosure agreements. Work closely with academic colleagues to help them produce valued publications that respect your company's need for confidentiality.
- Help clarify and articulate your company's position on longer-term professional training versus short-term skills acquisition and how this affects your relationship with educational institutions.

If you are in academia:

- Convince your colleagues of the efficacy of good SE practice. Show them that a project with good design, construction, quality assurance, and so on is better than a thrown-together project.
- Explain the implications and opportunities of accreditation and licensing and what effect they could have on expectations for a CS/SE degree. Teach an ethics module. Include information about the Software Engineering Coordination Committee (SWECC) and its projects.
- Integrate some SE into any software course you teach, and help your colleagues, especially those building large systems, inject some SE principles into their projects. Be sure to note any successes, however small.
- Have your SE project classes help the software efforts of non-SE colleagues.
- Include industrial SE practitioners as an advisory board and as guest lecturers.
- Advocate required industrial internships for students. Monitor the results and benefits.
- Take minisabbaticals in industry, even if the hosting corporation isn't funding it. Invite industry experts in for a sabbatical, a mini-sabbatical, or to be a "software artist-in-residence" to inspire and mentor students and staff.
- Develop and offer collaborative educational programs with colleagues in computer engineering, business management, or industrial engineering.

Too often, it disdains considering the gap between best and current practices—a significant education and research issue. Industry, on the other hand, is far too slow to adopt practices validated by research and experience or to invest in reducing the practices gap.

Society and industry have tolerated the consequent poor quality and practice because of the shortage of trained practitioners and the dearth of experienced managers who can recognize and sell good practice. The lack of a mature infrastructure and the influence of societal and business pressure make the widespread recognition and adoption of best practice slow and sporadic. Unfortunately, even if initial professional education more quickly tracked the emerging body of knowledge, many areas deemed critical would still be excluded. Internet-based e-commerce technology, for example, is moving so rapidly that only a handful of institutions have tried to offer it—even industry has a hard time keeping up.

Industrial-academic partnerships in education and research will enable practitioners to learn and hone a broader range of skills and practices. The efforts we have described will significantly drive the maturation of SE as a profession, but we will need to sustain and build on them to bring SE closer to its ultimate goal of respectability. ❖

Acknowledgments

We greatly appreciate the excellent suggestions by colleagues who reviewed an earlier draft of this article: Patricia Collins, Paula Hawthorn, Robert Kessler, Joe Podolsky, and Anthony Wasserman.

References

1. G. Ford and N.E. Gibbs, "A Mature Profession of Software Engineering," Tech. Report CMU/SEI-96-TR-004, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1996.
2. P. Bourque et al., "The Guide to the Software Engineering Body of Knowledge," *IEEE Software*, Nov./Dec. 1999, pp. 35-44.
3. D. Gotterbarn, "How the New Software Engineering Code of Ethics Affects You," *IEEE Software*, Nov./Dec. 1999, pp. 58-64.
4. G.L. Engel, "Program Criteria for Software Engineering Accreditation Programs," *IEEE Software*, Nov./Dec. 1999, pp. 31-34.
5. G. Pour and A. Hambaba, "An Undergraduate Software and Information Engineering Curriculum under Development at San Jose State University," *Proc. Frontiers in Education (FIE) Conf.*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. XX-XX.
6. D.L. Parnas, "Software Engineering Programs Are Not Computer Science Programs," *IEEE Software*, Nov./Dec. 1999, pp. 19-30.
7. M.J. Lutz and J.F. Naveda, "Crafting a Baccalaureate Program in Software Engineering," *Proc. 28th SIGCSE Tech. Symp. Computer Science Education*, ACM Press, New York, 1997, pp. XX-XX?

8. M. Jackson, "Will There Ever Be Software Engineering," *IEEE Software,* Jan./Feb. 1998, pp. 36-39.
9. A. Wasserman, "Software Processes and Software Professionals in the 21st Century," *Cutter IT J.*, Sept. 1999, pp. 17-23.
10. M.L. Griss, "Letter from the SIGSOFT Executive Committee," *ACM SIGSOFT, Software Eng. Notes*, Sept. 1998, pp. 1-2.
11. J.R. Speed, "What Do You Mean I Can't Call Myself a Software Engineer?" *IEEE Software*, Nov./Dec. 1999, pp. 45-50.
12. D.J. Bagert, "Texas Board Votes to License Software Engineers," *ACM SIGSOFT Software Eng. Notes*, Sept. 1998, p. 7.

***Gilda Pour** is a professor of software and information engineering at San Jose State University, where she helped develop a software and information engineering curriculum. She develops and teaches courses in object-oriented and component-based software engineering and distributed object computing in both industry and academia. Her industrial and research experience is in object-oriented component-based enterprise software engineering, with special emphasis on automated generation of Web-based enterprise applications. Pour received a PhD in computer science/software engineering from the University of Massachusetts. Contact her at gpour@email.sjsu.edu.*

***Martin L. Griss** is principal laboratory scientist for software engineering at Hewlett-Packard Laboratories, where he has researched software engineering processes and systems, systematic software reuse, object-oriented development, and component-based software engineering. He created and led the first HP corporate reuse program and participated in the development and execution of the HP corporate software initiative. Griss received a PhD in physics from the University of Illinois. He is an adjunct professor at the University of Utah and a member of the ACM SIGSOFT Executive Committee and SWEEP. Contact him at griss@hpl.hp.com.*

***Michael Lutz** is Motorola professor of software engineering at the Rochester Institute of Technology, where he heads RIT's undergraduate software engineering program. His interests in software architecture, software design, and lightweight formal methods led to development of core software engineering courses in these areas. Lutz earned an MS in computer science from the State University of New York at Buffalo. He is a member of the editorial board of* Computer *and of the IEEE CS Educational Activities Board. Contact him at mjlics@rit.edu.*