

ICT Professionalism¹

By David H. Gleason

Principal Consultant, ITforProgress.com

www.Linkedin.com/in/dgleason

Copyright © 2003

Rev. 11/13/09

Abstract:

This paper addresses issues of the ethics of professionalizing software engineering. While recent activities have impeded movement toward licensure in the United States, software engineering is likely to move inexorably toward legal authorization. This paper explores the ethics of professionalism, the state of licensure and certification efforts, and analyzes practical difficulties with software engineering. It then makes a series of recommendations on how to approach the professionalism issue, under the assumption that, sooner or later, formal authorization of software engineers is likely.

Introduction

Information and Communication Technology (ICT) development and implementation continues to be extremely problematic, and many observers have recommended a professionalization, even licensing of its practitioners. This paper argues that a move to software engineering professionalism and formal authorization is inevitable, and that careful attention to the parameters of professionalism is needed now.

The paper will discuss the concept of professionalism in general, then review the current status of ICT licensure. It will discuss practical problems with systems development and implementation, and explain how a “profession” of ICT might mitigate against these difficulties. It will conclude with a possible conjunction of business motives that could lead to professionalization of ICT in industry and formal authorization of software engineers.

The paper will use the modes of professionalism in medicine, law and electrical engineering to elucidate some of the benefits and risks of licensure, and discuss the work done so far by the Association of Computing Machinery (ACM) on the issue of professionalization. Unless otherwise specified, this paper refers to professionalizing activities in the United States.

1. The Concept of Professionalism

The word “professionalism” conjures images of well-trained, highly qualified and concerned individuals, working diligently to serve the needs of their clients. Be they doctors, lawyers or electrical engineers, professionals are thought to serve a higher calling than simply maximizing profit. With exceptions, we think of professionals as trustworthy by virtue of their titles.

There is good reason why this should be so. Trained and licensed professionals have the imprimatur of the community. Doctors must endure rigorous education, practical work and state examination before they earn the title of “licensed M.D.” American lawyers must be licensed in order to practice, and must subscribe to rules of conduct or risk sanction or

¹ This paper was presented at the ETHICOMP 2003 Conference in Lisbon, Portugal

expulsion. Electrical engineers are trained and licensed, and are held responsible to design and build safe generators and sub-stations.

The term “professional” may be most clear when we describe a state-licensed profession, like a licensed plumber. In the U.S., licenses may be granted by state government, which then has enforcement authority. It is the licensing agency that sets rules and determines whether a person is authorized to practice their profession. Licensing becomes a legal activity, with practitioners authorized and legitimized by the state.

In many cases associations serve as professional authorities, like the American Medical Association (AMA), which contribute to the knowledge, productivity and success of the practitioners. The American Bar Association (ABA) is a preeminent legal group that helps lawyers with learning, practice and advocacy.

Professions which are not licensed also have associations. The ACM contributes substantially to the knowledge and practice of ICT development. The difference is that lawyer members of the Bar are licensed by the state and systems engineer members of the ACM are not.

1.1. The down side

Neither licensing authorities nor professional associations can guarantee their practitioners to be ethical. Even after the rigors of medical licensure, doctors can be terribly damaging to their patients. Association with ACM does not prevent a developer from building malicious code. There is no silver bullet here.

The word “professional” itself carries some negative connotations. Professionals can be seen as aloof from practical problems, caught up in their own worlds; unable or unwilling to be accessible, for example, at the loss of a loved one due to disease or crime. Professionals, by virtue of identity with an elite group, may be insulated from criticism, and distant from their clients. Furthermore, authorizing agencies themselves are fallible, as when a destructive doctor is allowed to continue practicing or an examination is flawed. Indeed, associations may try to foist their views on others who already know better. This parochialism becomes particularly problematic in the international arena where associations are biased by cultural assumptions, for example, in the debate between western medicine and acupuncture.

Associations, like the AMA, which often contribute to the knowledge-base of a profession, are imperfect. Professional groups make mistakes, as when a medical treatment that is common practice, such as estrogen supplements for some post-menopausal women, turn out to be harmful.

Furthermore, professionals can fail society in many ways while looking out for their own interests, as with lawyers who care only about winning, and not about the truth. And professions can become corrupt, as when construction companies collude with municipal building inspectors to produce shoddy work.

The term “professionalism” itself is ambiguous. Originally stemming from profession of fidelity to a religious order, it grew to mean any “calling or occupation” from which a living is earned, and later, “engaged in one of the learned or skilled professions” [OED, 1973]. More recently, “profession” has come to mean licensure by a governing agency that authorizes practice of defined and constrained activities in exchange for remuneration.

However, the term is often used in situations that require no such association, as in a “professional painter,” “professional window-washer” or even “professional gambler” – any activity from which income is derived. While all the uses are technically correct, we mean something special when we call an activity a “profession.”

Licensing varies from state to state. Each state government has a licensing agency for physicians. It is through the agency’s professional standards that doctors are given license to practice. License carries with it a set of rules. For ethics violations, licensees can be sanctioned, or lose their right to practice.

In this framework, licensing becomes synonymous with profession. Licensing structures apply to doctors, lawyers, electrical engineers and, for that matter, electricians and plumbers. It is in this strict sense of the word that professionalism means something extraordinary: fidelity to a licensing authority with a code of conduct (ethics), the violation of which can result in a removal of licensure and a complete cessation of practice.

For the purposes of this paper, professionalism requires licensure by a governmental authority. Thus a “software professional” must meet a standard set by the state. It will become clear that this is a complicated problem. Like the early days of electrical engineering, associations exist, but their authority is limited, and membership is optional. By this definition, with a few exceptions, in the U.S. there is no such thing as a “professional software engineer.”

2. Towards ICT Professionalism

2.1. The status of licensing software engineers in the U.S.

Over the last 40 years a body of work on software engineering professionalism has emerged. The term “Software Engineer” has been in use since the 1960’s, and much effort has gone into thinking through the issues. In 1993 the Institute of Electrical and Electronics Engineers (IEEE) Computer Society (IEEE-CS) and the ACM formed a Steering Committee for the Establishment of Software Engineering as a Profession, superseded in 1998 by the Software Engineering Coordinating Committee (SWECC), to investigate professionalism among software engineers. In the summer of 2000 the ACM withdrew from the process, stating that:

“Our state of knowledge and practice in software engineering is too immature to warrant licensing. Moreover, Council felt licensing would be ineffective in providing assurances about software quality and reliability.

“At its meeting in May 2000, the Council further concluded that the framework of a licensed professional engineer, originally developed for civil engineers, does not match the professional industrial practice of software engineering. Such licensing practices would give false assurances of competence even if the body of knowledge were mature; and would preclude many of the most qualified software engineers from becoming licensed.” [ACM Summary, 2000]

While the ACM has withdrawn from the SWECC process, the IEEE continues work to develop a Software Engineering Body of Knowledge [SWEBOK, 2001]. This project is seen

as prerequisite to licensure, because a licensed discipline must have a defined and widely accepted knowledge base from which to educate and test aspirants. And although the ACM has withdrawn from the current process, they have not precluded the possibility of licensure in the future.

Even if a path to licensure remains uncertain, a strong start in the direction of ICT professionalism was accomplished with the adoption of the Software Engineering Code of Ethics and Professional Practice by the ACM and IEEE. The Code outlines an extensive set of parameters for professional comportment by ICT practitioners, including, for example, "Software engineers shall act consistently with the public interest.... Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.... Software engineers shall ensure that their products and related modifications meet the highest professional standards possible." [SECODE, 1999]. These rigorous practice constraints provide a guideline for any future movement toward licensure.

Similar to the code of ethics adopted the AMA, the ABA, and other professional associations, adherence to the code is a requirement of membership. Violations can result in revocation of membership.

Several attempts have been made to license "software engineers." In 1998, Texas produced one of the first pieces of legislation in the area requiring licensure of software engineers providing services directly to the public. While this attempt was a step toward professionalism, and started a vigorous debate, the effort was fraught with problems, among them the speed with which legislation was adopted. The Texas Board of Professional Engineers developed the following definition for the "practice of software engineering." The text is too vague and broad to be practically applicable:

"The practice of software engineering will mean a service or creative work such as analysis, design, or implementation of software systems, the adequate performance of which requires appropriate education, training or experience...." [Adams 1999]

As of this writing, Texas still has no licensure exam, requiring instead an experience waiver. Because the law was so quickly adopted, and the Body of Knowledge is not yet defined, it is not possible to develop a meaningful exam.

The state of licensure for software engineers, and a true "professionalism" as it is defined herein, is a work in progress. In the U.S., no substantial licensing process is underway, and the Texas results are mixed. Furthermore, the understanding of what it means to be a "professional software engineer" remains unclear. In the next section we turn to the practical problems of defining ICT professionalism

3. Challenges to ICT Professional Practice

In actual practice, software engineers are faced with a dizzying array of issues in a rapidly changing environment. Because of this, it is not surprising that the ACM deemed it premature to define a body of knowledge. The range of knowledge that software engineers must have is enormous. They face a range of challenges, including:

3.1. Customization

More and more companies are implementing customized off-the-shelf applications, rather than developing applications from scratch. Nevertheless, customization itself can cost millions of dollars, requiring clear specifications and experienced, attentive developers who are familiar not only with generalized techniques but also with the application at hand. For example, Customer Relationship Management (CRM) solutions like Seibel are complex, and require knowledge and experience, as well as project management and technical expertise, for effective implementation.

3.2. Complexity of integrating many systems

ICT systems are comprised of arrays of hardware running millions of lines of software code. While a given configuration may be reliable, most businesses require integration of their disparate systems. For example, within leading organizations, current ICT projects may attempt to integrate Enterprise Resource Planning (ERP), CRM, integrated e-mail, voice communications, document management, supply chain management, sales, partner integration, real-time web sites, transportation, human resources, marketing, surveillance, security, streaming video, and so on.

While such integration makes sense intellectually, in practice it is all but impossible. Some of the reasons it is so difficult include: different operating platforms of various systems, complexities of data integration and unreliable code. But the fact that no one knows how all the systems work is the most difficult problem. Indeed there is no way that anyone could. Computer systems are simply too complex.

Microsoft Windows XP has more than 50 million lines of code. No one knows them all, or how they work together. Windows, and all large coding efforts, resulted from the collaborative effort of many developers. Communication among colleagues is the trick, not full knowledge by any of them.

Furthermore, software engineers must also understand the business environment and real-world requirements of the systems they are developing. They are not working in scientific sterility on precise experiments, but rather in the messy world of human life. Like doctors, they must diagnose the needs of their patients, be they human or mechanical, and prescribe solutions. Unlike doctors, they get involved in every aspect of commercial activity.

On top of all this practical complexity, mathematical complexity says that the more components there are in a system, and the more people involved in its development, the more likely a project and system will produce unpredictable results.

3.3. Lack of tools and too many competing development methodologies

There are many competing methodologies on the market for software development, from proprietary methods from large firms, to Yourdon, to “Extreme Programming.” Each has its merits, but a developer could spend years getting to know the subtleties of a single one. When a new development team comes together, they may have very different approaches.

Unlike engineering, where blueprints and drawings have very specific rubrics, there is no commonly accepted tool for communicating software requirements. There are popular but

incomplete tools, and there are different schools of thought about how to structure specification documents, such as use cases, screen shots and flowcharts.

Moreover, there exists no standard contract for software development services, as there is in architecture, for example. This makes litigation extremely difficult in problem projects. It is often impossible to really know who is at fault for software failures.

3.4. Known ICT project risks:

Subsumed coding faults result in accumulated errors in complex IT systems. There are always risks of failure, including systems, project, and even business. Financial loss arises due to unforeseen problems resulting in overruns or unreliable systems. And because of the overall complexity and haphazard systems development, there is always the risk of unpredictable system behaviors.

3.5. Obsolescence of equipment, software and knowledge

The knowledge base for developers changes substantially every 36 months. While companies like Microsoft, Cisco, Sun and many others offer certification, incentives are usually based on selling software licenses and upgrades.

Sales and the evolution of the industry is exciting and leads to ever more-powerful systems, however, organizations are often pressed to apply cutting-edge technology where more established systems would work better for less money.

Rapid obsolescence, combined with sales pressure leads to systems that are very dynamic, make the body of knowledge for software professionals yet more elusive.

3.6. Organizational change

Organizations undergo major operational changes with the implementation of ICT systems. Processes that have been in place for years are altered. There is no easy way to minimize the impact on staff, other than to plan thoroughly, identify stakeholders broadly and evaluate project impacts, work through problems as they arise and to vigorously communicate on open issues until they are resolved. It is the nature of information systems that adoption requires training, and changing systems can be difficult for workers.

The ICT adoption issues that organizations must work through are in part technical, but adoption also requires changing operating processes without disrupting the flow of business. It is like surgery: the patient must be kept alive and healthy during the operation. Often, companies do not pay enough attention to how much disruption IT implementations might cause. Professionals need to figure out how to mitigate the negative impact as much as possible throughout systems development and implementation. This mitigation work requires a great breadth of knowledge that gets into human resources and business management.

3.7. Antagonism between management and ICT

In the U.S., an embattled atmosphere pervades ICT departments. It is difficult enough to keep systems up and running, much less respond to the changing and unclear needs of the business. Moreover, management and IT often speak different languages, and simply have difficulty understanding one another. There is financial pressure with limited ICT knowledge

on the management side, combined with problematic motives on the IT side: coders want to code, and they are motivated to learn new tools and techniques. Discipline is not strong. Meanwhile, executives need alignment, but the lack of standards for communication make it difficult to turn business requirements into practical software applications.

4. How Professionalism Might Mitigate these Difficulties

Because of the above challenges, and many others, professionalizing the software engineering industry is a problem. It may be hard to know where to start, much less to commit to a timeline for the process. And yet, software failures continue unabated, and the National Institute of Standards and Technology (NIST) has recently reported that software bugs cost U.S. users \$59.5 billion per year. Something needs to be done. What follows is a high-level road map for how an effort might proceed. The map focuses on strategies to mitigate the difficulties outlined above.

4.1. Start with project management

Recognize that software development is project-oriented, and begin with the tools and techniques that have been established by the Project Management Institute. Enhance this with the practices outlined in the CMM.

4.2. Use the Software Engineering Code of Ethics as a framework

The Software Engineering Code of Ethics and Professional Practice, as adopted by the ACM and the IEEE is a major accomplishment in professionalizing the industry. It codifies the terms under which a software engineer should behave professionally, and it was adopted through a rigorous international process that garnered support from a wide range of academic and professional people.

4.3. Use the work to date by ACM and IEEE

The work done by the Software Engineering Coordinating Committee is basis material. Before it abandoned the process, the ACM wrote a series of critical documents, the objections in which should be taken and used to improve the professionalization process. Additionally, the Software Engineering Body of Knowledge [SWEBOK, 2001] as it stands is a massive compendium and an excellent draft for further development. Together, these three initiatives supply enormous leverage for movement forward.

The ACM has argued that licensure would not improve software quality, and that it would exclude qualified participants. It is the position of this author that these questions are up for debate, and that formal authorization, either through an association or licensure, if done correctly, could indeed improve the quality and reliability of software. While parallels to mechanical and electrical engineering may be imperfect, professionalism in these practices is a requirement for a safe physical infrastructure.

4.4. Learnings from Texas, the international community and the history of licensed professions

The experience of Texas, licensed professions and nations that have adopted licensing procedures should be used to inform further attempts to professionalize the IT industry. The

successes and failures of these initiatives can provide key insights to facilitate the process moving forward.

The history of licensure of electrical engineers, and many other disciplines, is of particular use. The process starts with professional association and evolves into licensure as practitioners and the public realize the dangers of incompetent work. Analysis of the shifts from small groups gathered to build a profession to the legal and political challenges of crafting licensure law can help the software engineering community organize for maximal success.

4.5. Adopt a stakeholder-oriented software development methodology

On the horizon, stakeholder-oriented software development holds promise for improving professionalism in the industry. In order to succeed at a basic level, IT must align with business needs and be timely and affordable. To excel, it must have a positive effect on stakeholders.

The costs of subsumed errors in software design and development increase as projects progress. By planning out the impact of activities and deliverables on stakeholders, unforeseen risks and problems can be identified and mitigated as early as possible. Such a procedure reduces project risks, allowing limited resources to be deployed where they are most needed, improving quality in the long run.

Even though techniques are available to bring software project management to a level of reliability where expectations and specifications can be consistently met (e.g., the Capability Maturity Model [CMM] from the Software Engineering Institute at Carnegie Mellon), most software projects today take place at the ad-hoc level. Once organizations achieve higher levels of the CMM, the next change needed for widespread improvement of project outcomes will be rigorous stakeholder impact analysis.

The analysis process can be facilitated by a tool known as a “Software Development Impact Statement (SoDIS),” developed by Don Gotterbarn and Simon Rogerson. SoDIS uses the work breakdown structure of a project, identified stakeholders and a set of questions to identify unforeseen project risks.

4.6. Matrix

The following table shows where the strategies above mitigate the difficulties outlined in section 3. Checks indicate areas where the mitigation strategy can be effective at reducing the difficulty:

Mitigation \ Difficulty	Project Management	Code of Ethics	ACM & IEEE Groundwork	Historic Experience	Stakeholder Analysis
<i>Customization</i>	✓	✓		✓	✓
<i>Complexity</i>	✓	✓			✓
<i>Lack of Tools</i>	✓		✓	✓	
<i>Project Risks</i>	✓	✓	✓	✓	✓
<i>Obsolescence</i>			✓		
<i>Org. Change</i>	✓			✓	✓
<i>Antagonism</i>	✓	✓		✓	✓

5. Business motives - How Professionalization Mitigates Risk

5.1. Why should businesses push for professionalization?

Business is interested in software professionalism because they are losing so much money and profit due to a lack of standards and professional practice by developers and IT experts. As cited earlier, NIST has established that \$59.5 billion is wasted annually due to software bugs. Business have a very high interest in improvement to software quality and the responsibility of developers. Licensed practitioners have specific skills, define a profession, and carry with them the expectation of professional, reliable practice. Licensure requires infrastructure that can be used to resolve legal disputes, and in the best case, defined recourse to litigation can assure buyers that they will get what they need.

Many business leaders believe that professionalism could lead to ICT projects with a more predictable return on investment. Project management techniques are finding their way to the mainstream, and would need to be incorporated into the accepted body of knowledge. Project management practice includes processes for communication, project control, risk mitigation, among others. These widely used practices could dramatically improve software project outcomes if they were consistently and rigorously applied.

The trust between developers and businesses has eroded, due to the antagonism explained above. Increased professionalism and, in particular, devotion to the needs of the stakeholders, can improve trust.

Better satisfaction among workers results when ICT systems are working effectively. For example, if a call center worker receives a call and needs to wait 30 seconds for the customer information to populate the screen they will, after just a few days, become exasperated with their work.

A move toward professional authorizations, supported by commonly accepted methods and a standard contract, would go a long way toward improving software project outcomes. Organizations like the ACM, IEEE, legal authorities and government will need to collaborate to realize these goals.

6. The Balance Needed

The road to software professionalism is challenging. There should be no delusion that the outcome will necessarily be good. The potential for corruption in licensure, inspection and permitting is significant. The licensure process could be too arduous and discourage good applicants. Licensure could squelch creativity, and harm business. Perhaps most daunting, licensure could greatly increase the cost of software development, already an expensive proposition.

And yet, inevitably, software on which people's lives depend needs to be of a higher integrity than can be assured through certification by private companies. If not by licensing, then how will the ability of software engineers be validated, their authority legitimized?

At some point, lawyers and politicians will become involved in this process. If the industry can come up with the parameters it would use for licensing before the issues enter the courts, there will be more time for careful reflection and planning. Once the issue does move to the legal system, the domain of information ethics devoted to software development will change forever. Now is the time for constructive thought.

There are a number of critical open questions, such as:

- How do we build information ethics insights into the legal move toward licensure?
- What is needed as the legal basis of software engineering, such as a standard contract and specification model, in order to hold all parties accountable?
- Who in the legal system do we want to codify the licensing of the software engineering profession? How should associations, academics, practitioners and lawyers work together to develop a program that will ultimately result in better software?
- What can we learn from existing licensing processes and history, particularly in mechanical and electrical engineering, that can help us to avoid pitfalls?

Licensing may be an imperfect solution, but there is no doubt that we want our bridges built by licensed professionals

7. Conclusion

This paper has discussed the issues of software engineering professionalism, but a great deal of thoughtful dialog, process and inclusiveness is needed in order to set the stage for sensible outcomes.

The debate will move inexorably toward professional authorization because software bugs cost \$59.5 billion a year and because, more and more, we trust our lives to software-controlled systems. But the parameters of moving forward must be reviewed with great care. If licensure is to come about, it must take into account, for example, both the IEEE SWEBOK and the objections of the ACM. The professionalizing process needs to be rational, deliberate, inclusive and practical in order to be of utmost value.

8. References

- ACM (2000), Summary of the ACM position on Software Engineering as a licensed Profession, online at http://www.acm.org/serving/se_policy/selep_main.html/ accessed 7/26/02.
- ACM Panel on Professional Licensing in Software Engineering Report to Council (May 15, 1999) online at http://www.acm.org/serving/se_policy/report.html/ accessed 7/27/02
- Adams, Kenneth R. (1999), Texas Board of Professional Engineers to License Software Engineers, online at <http://www.denveritp.org/legislative/certification.html> accessed 7/26/02
- Bagert, Donald J. (1999) The Licensing of Software Engineers and its Effect on Certification, online at <http://www.isacc.com/isacc99/presentations/bagert/sld001.htm>, accessed 7/26/02
- Gotterbarn, Donald, Software Development Impact Statement (1999) online at <http://www.ccsr.cse.dmu.ac.uk/conferences/ccsrconf/abstracts99/gotterbarn.html>, accessed 7/29/02
- IEEE Certification (1999), online at <http://www.computer.org/certification/> accessed 7/27/02
- Knight, John et al (2001) On Licensing Software Engineers Working on Safety-Critical Software, online at http://www.acm.org/serving/se_policy/safety_critical.pdf accessed 7/26/02
- Mead, Nancy (2002) Issues in Licensing and Certification of Software Engineers, online at <http://www.sei.cmu.edu/staff/nrm/license.html/> accessed 7/27/02
- Notkin, David et al (2000), An Assessment of Software Engineering Body of Knowledge Efforts, online at http://www.acm.org/serving/se_policy/bok_assessment.pdf, accessed 7/26/02
- SECODE (1999), Software Engineering Code of Ethics, online at <http://www.computer.org/tab/seprof/code.htm/> accessed 7/26/02
- Shorter Oxford English Dictionary, 1973
- SWEBOK website (2001), online at <http://www.swebok.org/> accessed 7/26/02
- Texas Board of Professional Engineers, online at <http://www.tbpe.state.tx.us/> accessed 7/27/02
- Thompson, J. Barrie, "Slowly Reaching Towards Software Engineering Professionalism: A Report on Recent Activities Across the World." ETHICOMP 2001 Proceedings, V.1, pp 14-21.