EDITOR PHIL LAPLANTE
Penn State, plaplante@psu.edu

A Brief History of Software Professionalism and the Way Forward

Phil Laplante, Pennsylvania State University

A brief history of professionalization in software engineering, particularly in the United States, is given. The current situation and recommendations going forward are then offered.

Software is pervasive, including ways in which it can place our lives at risk (for example, flight, railway, and traffic control). But even noncritical systems can have unplanned, ad hoc interactions with critical ones, increasing risk exposure. Society expects a standard of competence, professionalism, and accountability from its doctors, nurses, and other professionals who hold lives in trust. Yet anyone can write software that can appear in or interact with critical systems, so what does "software professional" mean, and what are society's expectations for those individuals? The IEEE Computer Society (CS), other professional societies, and scholars have tackled this problem for decades. I'd like to recount some of those efforts to place into perspective a proposal for how society must proceed to ensure an appropriate level of professionalism, especially from those working on software for critical systems.

Digital Object Identifier 10.1109/MC.2020.3004017
Date of current version: 4 September 2020

STANDARDIZING CURRICULA

In the early 1950s through the mid-1960s, software systems were significantly smaller and simpler by current standards. There were a few critical software systems, for example, in flight control, but they were not ubiquitous, and the public's general perception of computers was that of mysterious devices used in universities and government think tanks. Few thought that software directly affected their lives or put them at risk (making the 1964 film *Fail Safe*, in which a computer error caused a nuclear war, more terrifying.) Professional programmers, as they were often called, worked individually or in small teams and had very diverse educational backgrounds, but they never came from any "computing" or "software" degree programs, which didn't exist.

Anyone can write software that can appear in or interact with critical systems, so what does "software professional" mean, and what are society's expectations for those individuals?

A turning point occurred when more than 50 such professionals attended a 1968 NATO conference on software engineering to address "software, either as users, manufacturers, or teachers at universities," leading to early efforts to standardize computing curricula and create a profession. Many computer science and programming curricula emerged, but they were wildly diverse. The need to address the inconsistency of these curricula led the Association for Computing Machinery (ACM) to develop its "Curriculum 78" on undergraduate computer science/programming, which prescribed both program and individual course structure and details.[1]

In a 1980 article, Harlan Mills[2] distinguished "software engineering" from programming. Mills noted that "the effective practice of software engineering must be based on its technical foundations just as any other engineering activity, in combining real world needs and technical possibilities into practical designs and systems." By "real world needs," I think he was implying that those building the systems should have domain expertise.

In 1984, the Computer Science Accrediting Board (CSAB) was formed to accredit academic programs, and in 1988, a joint ACM/IEEE task force created a new model for computing curricula. Over the next several years, more computing programs sprang up worldwide. The CSAB became the authoritative body for computing programs and a member of the Accrediting Body for Engineering and Technology, which, among other things, provides criteria for certifying academic engineering programs to license a Professional Engineer (PE). All these efforts contributed to an elevated and more uniform level of education for computing professionals, including software engineers.

SWEBOK AND CSDP

In 1993, the ACM/IEEE Steering Committee for the Establishment of Software Engineering as a Profession recommended the following:

- Adopt standard definitions
- Define a required body of knowledge and recommended practices
- Define ethical standards
- Define educational curricula[3]

as a means toward greater uniformity and professionalism in software engineering. (I will save the discussion of ethical standards for a future column.) By 1995, the steering committee was considering licensing and certification issues.[3] These recommendations, and an extensive collaborative project over many years, led to the creation of the first Software Engineering Body of Knowledge (SWEBOK).

In 2002, the CS introduced the Certified Software Development Professional (CSDP) designation, with testing built on top of SWEBOK. While corporations had been offering various software certifications for years (generally, as proficiency in some proprietary software), the CSDP was a vendor- (and application domain)-agnostic certification of software engineering competency. The CSDP and SWEBOK continued to be refreshed over the years, but, in late 2014, the CSDP was discontinued, though all issued certificates were capable of being converted to new, comparable ones offered by the CS. Currently, these certifications include the Professional Software Engineering Master (PSEM) and Professional Software Engineering Process Master (PSEPM). An Associate Software Developer certification is also offered by the CS and is very popular among students and individuals with less software experience than a PSEM would require.

PROFESSIONAL SOFTWARE ENGINEER

In 1998, the Texas Board of Professional Engineers licensed Don Bagert as the first PE in software engineering in the United States. A few dozen others later became licensed in Texas. In 2008, the Software Engineering Licensure Consortium (SELC) launched the development of the Principles and Practice (P&P) of Software Engineering exam, which was needed to complete the pathway to licensing professional software engineers in other U.S. states. Having been a long-time licensed electrical engineer (with embedded systems software experience) and recently having earned the CSDP designation, I was chosen to create the team and lead the process to develop and maintain the P&P of Software Engineering exam. The task of leading this development was technically, logistically, and politically very challenging, and it consumed a significant part of six years of my life.

The SELC was led by IEEE-USA, through its professional licensure committee, which supported the effort financially and administratively. The CS was also a financial supporter and member of the SELC, along with the National Society of Professional

Engineers (NSPE) and Texas Board of Professional Engineers. The National Council of Examiners of Engineers and Surveyors (NCEES) (this is the same entity that develops all engineering licensing exams in the United States) provided in-kind staff, expertise, and logistical support for the intensive exam development and maintenance process.

The CS support of the software PE exam was largely predicated on the expectation that SWEBOK would be used as the basis of the licensing exam body of knowledge (BoK) and that possibly CSDP review courses or certification would hold some weight in the initial licensing or at least ongoing licensing requirements. Unfortunately, the NCEES rules for the examination required development of a very specific BoK using its standard process, so neither SWEBOK nor the CSDP could be used as the CS intended. The details of the development of the BoK and exam can be found in Laplante et al.[4]

In 2013, the test was made available in 30 states and then in 40 states the following year. By 2015, only a few dozen new software engineers had been licensed in the various states, and by 2018, there were still fewer than 60.

Financial support for the effort slipped. The CS had never intended to support the effort indefinitely (only to help achieve liftoff) and, eventually, reduced and then eliminated its funding. Then, a significant blow occurred when the IEEE-USA board, which had been one of the prime movers in the SELC and had provided more than 50% of the financial support over the years, voted to eliminate all funding. With financial support only from the Texas Board and NSPE and with few taking the exam, the future looked grim. By April 2019, the P&P exam for software engineers was discontinued by the NCEES due to the lack of examinees and funding, effectively ending the possibility of licensing new software engineers in the United States.

The licensing of software engineers for critical systems was quite controversial—there were many strongly worded articles and editorials in support or opposition to licensure. While chairing the exam committee, I received some emails in support and several nasty ones in opposition. Even I had been in opposition to licensing software engineers before supporting it, I had thought it was too soon to do so.[5] There were also thorny issues with the reciprocity of licensing between states and countries, industrial exemption questions, grandfathering, and more that were never fully sorted out.

But it was very disappointing to have been involved in an intense effort to define the need for licensing certain software engineers, create the pathway, and then see so few willing to take the exam or be able to meet the qualifications, followed by the collapse of support. I am convinced licensing software engineers in any way will never happen again (at least not in the United States). So how can we ensure competency and professionalism of software engineers working on certain critical systems?

Mills noted that "the effective practice of software engineering must be based on its technical foundations just as any other engineering activity, in combining real world needs and technical possibilities into practical designs and systems."

A WAY FORWARD?

Since the demise of the path for licensing professional software engineers in the United States, I have been struggling with an appropriate way forward for ensuring that those working on critical software systems have the suitable knowledge and skills and are trustworthy and accountable. At first, I thought the solution was a "libertarian" one. Ask all software engineers working on critical infrastructure to swear to an appropriate code of ethics and demonstrate applicable education and experience and training. Leave it up to practitioners to follow the honor system, employers to police it, and civil courts to enforce any transgressions. But I later dismissed this plan because I am distrustful of all the players just mentioned.

Here is my current thinking. For critical infrastructure, I believe the software professional (or engineer) must demonstrate sufficient education, competency, currency, and accountability of such in both the domain discipline (of the application) and software engineering. The software engineer needs to know what he or she doesn't know about the domain, and the only way to approach that is to have substantial domain knowledge. For example, one can be an "excellent" software engineer but have very little knowledge of avionics. A software engineer simply writing flight-control software based on an algorithm given to him by an aerospace engineer is not enough—there are too many domain nuances that are often the source of critical faults.[6]

While software engineering general knowledge and skills are portable across domains, domain knowledge is not. So one might even be an engineer with competency in software and have a good understanding of avionics but not of medical devices. Thus, if you are going to work in two or more critical domains, you need to demonstrate competency in software and in all critical domains in which you are going to work. Having earned a degree in either the domain discipline or software engineering could serve one purpose or the other in contributing toward that credibility. But I think it is better for the relevant custodians of critical infrastructure systems to enforce the requirements for domain expertise accordingly. For example, in the United States, the Federal Aviation Administration (FAA) would establish criteria for avionics systems, the Nuclear Regulatory Commission (NRC) for nuclear power generation and distribution, and the Food and Drug Administration (FDA) for medical devices. These entities might keep a registry of such competent and accountable individuals.

The critical systems software engineer must also be competent, current, and accountable in the principles and practice of software engineering. This truth holds both for software engineers with domain competency and for domain experts who write software, potentially with no formal training in software engineering. For software engineering, the PE license was supposed to demonstrate competency and enforce accountability for software engineers. With the demise of the PE license, we are left with alternative mechanisms for demonstrating competency in software engineering. The PSEM or PSEPM, along with an appropriate education and experience, can do that. Graduate degrees in software engineering can demonstrate that, but again, these proof points can be determined by the customer, whether it's the FAA, NRC, FDA, or their international and commercial equivalents.

And what about experience alone? What about that person writing flight software with no formal training in either software or avionics? Experience is great, but it is not necessarily proof of mastery, expertise, or even competency. That's why the standardized testing, vetting of experience, and refreshment and revalidation of knowledge and skills required for PE licensure and certain certifications (such as PSEM) are essential. The domain custodians can set these standards and police them. If a person claims on a resume that he or she has worked on such-and-such a system, even if true, it does not demonstrate what contributions that individual made to the project. Even background checking is unreliable. So having a license, certification, or pledge of ethics, which puts someone at risk for violating those principles (for example, loss of membership, fines, and even imprisonment in the case of real injury) helps provide some level of assurance to the public.

Standardized testing, vetting of experience, and refreshment and revalidation of knowledge and skills required for PE licensure and certain certifications are essential.

I am not saying this is the only way forward, nor have I worked out all the details. But it's one approach that's worth considering.

And what about every other person writing software and inserting it into the public domain in some way? That's a different problem for another column.

Speaking of which, I invite contributions for this column that are interesting, informative, possibly controversial, vendor agnostic, and accessible (to all readers of *Computer*). Ping me if you think you have an idea that could meet these criteria. C

REFERENCES

1. P. Nauer, B. Randell, and F. L. Bauer, *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. Brussels, Belgium: Scientific Affairs Division, NATO, 1969.
2. H. D. Mills, "Software engineering education," *Proc. IEEE*, vol. 68, no. 9, pp. 1158–1162, 1980. doi: 10.1109/PROC.1980.11814.
3. N. R. Mead, "Issues in licensing and certification of software engineers," in *Proc. 10th Conf. Software Engineering Education and Training*, 1997, pp. 150–160. doi: 10.1109/SEDC.1997.592449.
4. P. A. Laplante, B. Kalinowski, and M. Thornton, "A principles and practices exam specification to support software engineering licensure in the United States of America," *Softw. Qual. Prof.*, vol. 15, no. 1, pp. 4–15, Jan. 2013.
5. P. A. Laplante, "Professional licensing and the social transformation of software engineers," *Technol. Soc.*, vol. 24, no. 2, pp. 40–45, Summer 2005. doi: 10.1109/MTAS.2005.1442380.
6. E. Wong, X. Li, and P. Laplante, "Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures," *J. Syst. Softw.*, vol. 133, pp. 68–94, Oct. 2017. doi: 10.1016/j.jss.2017.06.069.

PHIL LAPLANTE is a professor at Pennsylvania State University. Contact him at plaplante@psu.edu.