When does software affect the health, safety and welfare of the public?

Phil Laplante
Penn State
January 2012

Abstract

Licensure of certain software engineers in the United States will be required in at least 10 states by 2013 and, likely, by all US states and jurisdictions (e.g. Puerto Rico) within a few years. The purpose of licensure is to ensure that those who offer software engineering services to the public are minimally competent. But which software engineers will need to be licensed? What kinds of kinds of software systems affect the health, safety and welfare of the public? The answers to these two questions are both a matter of law and of science. This paper addresses the scientific aspects of these two questions from the perspective of reliability engineering.

Introduction

States require licensure of certain civil, electrical, structural, and other engineers to ensure that any practitioner is at least minimally competent. The intent of licensure is to protect the public from injurious consequences of incompetent "engineers." Generally speaking, licensure is required if the engineer is involved in building a system whose failure could cause harm and is offering his services directly to the public (e.g. as a limited liability corporation) and not through a corporation, or government entity, which would assume the responsibility for the engineer's work

Currently, only Texas requires licensure for those working on systems that affect the "health, welfare or safety" of the public [1]. But by 2013, the states of Alabama, Delaware, Florida, Michigan, Missouri, New Mexico, New York, North Carolina, Texas and Virginia will require licensure for certain software engineers and more states will follow suit [2]. I believe that all U.S. states and jurisdictions will adopt some form of professional licensure for software engineers. I feel confident in this prediction because economic forces will make it so. When a software product fails in a way that causes injury, and that software was produced in a state that does not require licensure, plaintiffs' lawyers will ask "why did the state not do its duty to protect the public, as is done in other states?" And then the legislature of the state in question will act to require licensure.

Some philosophers worry that Software Engineering may never be regarded as true engineering [3]. Only a few years ago I, too, doubted that the science could come together to lead to a truly foundational software engineering discipline [4]. After I read Dennis Frailey's article [5], however, I become convinced that moving towards licensure was an important step towards establishing the foundational science needed. But whether philosophers consider software engineering as engineering or not is irrelevant to the issue of licensure. After all, states license all kind of non-engineering occupations such as barbers, plumbers, and tattoo artists in order to protect the public.

Licensure is generally obtained by earning an appropriate engineering degree, having several years of relevant experience, and passing two comprehensive exams – one a general test of engineering principles

and the other a test of the principles and practices of software engineering. It is beyond the scope of the article to discuss the nature of these exams, but this information will be made available to those who wish to take the examination through the National Council of Examiners of Engineers and Surveyors, the non-profit organization that is responsible for administering the licensure exams.

The laws setting forth guidelines for licensure of software engineers have yet to be written for most states. Hence, no one knows for sure which professionals will need to be licensed and which will not. However, it is known that state laws tend to require licensure for those offering services directly to the public (a matter that I have already discussed) and who work on those systems that can affect health, safety and welfare. Let's examine some hypothetical examples of the kinds of systems that could fall into this category.

What kinds of systems would affect the health, safety and welfare of the public?

Table 1 provides a sampling of systems that I think would require a licensed professional engineer if the system was being offered directly to the public without the protection of an industrial exemption.

Application Domain	**Individual**	**More than one person**
Commerce	vending machine	e-commerce site
Entertainment	tattoo machine	roller coaster
Financial	tax preparation software	pension fund management
Medical	implantable insulin pump	telemetry monitoring
Transportation	motorcycle	automobile
Utility	tanning bed	water treatment plant

Table 1: Some sample systems that affect the health, safety and welfare of the public.

I am not suggesting that Table 1 represents a comprehensive taxonomy of "licensable systems." Rather, the table represents a starting point for future discussions. Let's discuss each of these systems briefly.

Vending machine. Consider a software controlled vending machine that bakes refrigerated pizza slices on demand (these systems are popular in Europe). There are potential dangers in the functioning of the baking oven (potential for explosion or electrocution) as well as the potential for contamination if the refrigeration system fails. Such a system differs, say from a simple electromechanical candy vending machine, even if it is under some kind of software control (e.g. for computing change to be made). In fact, there is a potential for the change making software to fail somehow, short changing the customer for example. But this eventuality does not pose significant financial harm, and therefore, would be exempt from a licensure provision.

Tatoo machine. It was mentioned that tattoo artists must be licensed in most states. But what about a software-controlled tattoo machine? Clearly since the machine is invasive (i.e. it sticks needles in a human) it would fall under the licensure provision because a gross malfunction could cause serious bodily harm. But what would be the liability to the engineer designing the machine if the a software failure causes it to erroneously tattoo "Jean," the name of "Fred's" ex-wife on his arm, instead of his current

girlfriend, “Joan,” causing Joan to go into a rage and kill Fred? The answer is unclear, I think, and I will discuss the interactions of systems subsequently.

Tax preparation software. A malfunction in this software could cost significant financial loss to a taxpayer or taxpayers, even expose them to criminal prosecution and loss of liberty.

Implantable insulin pump. Clearly, a software malfunction in such a device could cause significant harm, even death. I do not know how FDA regulations would impose further constraints on the engineering of the system.

Motorcycle. The software controlling the cruise control system could fail somehow, causing the vehicle to crash. Clearly, there is the potential for harm to a human rider and nearby pedestrians. The same situation could occur for an automobile.

Tanning bed. If software controls the tanning lights and is capable of accidentally locking the bed closed and trapping an occupant, then significant harm could result.

e-commerce site. If poor software engineering practices lead to inadequate security provisions then customer account data, including credit card numbers and other identifying information, could be compromised.

Roller coaster. Terrible things could happen if some sort of software malfunction in the control system causes the car to achieve excessive speeds and run off the rails.

Pension fund management. Again, imagine that poor software security provisions allow an intruder to wipe out the retirement funds of thousands of persons.

Telemetry monitoring. Errors in processing, triaging, or displaying signals carrying vital signs of patients could lead to treatment errors and injury or death.

Water treatment plant. Inadequate security provisions (famously, vulnerabilities in the SCADA control software) could lead to an attack on the system that could make the system unsafe (e.g. inadequate purification) and present significant health hazards to the public.

Instead of trying to categorize every type of system that could affect the health, safety and welfare of the public, it is simpler to create a set of questions that can be used to determine if a software system can have an adverse effect on the public. Fortunately, such a set of questions exists – the Laplante-Thornton criteria [2]. These are:

1. Does the software control a device or devices that could directly inflict harm to a human being if there was a malfunction?

2. Does the software put the assets of an individual or corporate entity at risk beyond the normal amount of risk assumed in everyday business transactions?

3. Does the software expose identifying information of an individual or a corporate entity that would violate any federal, state or local laws (e.g. HIPPA, FERPA)?

4. Does the software interact with other systems in way that directly satisfies 1-3 above?

I contend that if the answer to any of these questions is "yes" then the software system, or the parts of the system would likely have to be created under the responsible charge of a licensed professional software engineer [2]. For example, the vending machine, tattoo machine, insulin pump, motorcycle, automobile, tanning bed, roller coaster, telemetry monitor, and water treatment plant all would answer affirmatively to question 1. The tax preparation software, e-commerce site, and pension fund management system would answer affirmatively to question 2. The tax preparation software and pension fund management and possibly the e-commerce systems might also answer affirmatively to question 3, if adequate precautions to protect personal information were not taken.

Systems interactions

What about question number 4, that is, a chain of interactions starting with a seemingly innocuous piece of software but eventually causing the catastrophic failure of some system that can harm the public? Do we need to consider all software and the interactions between software components within a system and between systems in order to have some sort of transitive closure of safety? For example, if a security breach to a "non-critical" system linked to a critical one causes a public disaster, should it be concluded that the 'non-critical" system was really "critical." The answer to this question, and others like it, are unclear and likely will need to be resolved in a court of law on a case by case or class basis [2]. But it is worthwhile to try to offer some guidance in this regard.

Consider a system of systems $S_1,\ldots, S_n$, $n\geq 2$. Suppose that each S_i, $2\leq i \leq n-1$interacts with S_{i-1} and S_{i+1} but that only S_1 interacts directly with humans (Figure 1). Suppose, however, that through a sequence of systems interactions a software failure in S_n causes a cascade of failures to S_1, which causes injury to some human interacting with S_1. What is the responsibility of the designers of S_n to the injured parties?

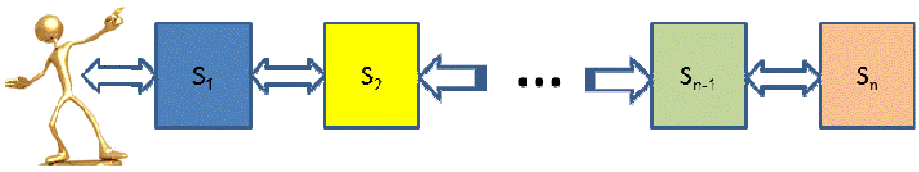

Figure 1 A system of systems. Only system S_1 interacts with humans.

While this question could only be answered by a jury, and on a case by case basis, I contend that the responsibility to the engineer should be reduced as a function of the distance of interaction. For example, while the engineer working on system S_1 should be 100% responsible for the consequences of a failure in S_1 that harms the public, the engineer for system S_2 would bear some responsibility, let's say one half, for a failure in S_1 if said failure could be shown to be due to a fault in S_2. Hence, the responsibility for the

engineer of system S_n, financial or otherwise, would be limited to $1/2^n$ of the total liability. Of course, the situation get more complicated based on the sequence in which the systems are developed, whether interactions where envisioned previously, whether standards based design is used and so on. And the simple model above only considers sequential interactions – what about a web of interactions? Clearly each case has to be considered separately.

Another question I am frequently asked is "what about software components that are produced in other countries, or in open source communities, or in states where licensure is not required? How does licensure apply in these situations? In this respect, software engineering is no different than other engineering disciplines in using other third-party furnished components. For example, when licensed civil engineers use steel produced in another country to build a bridge in the US, they are certifying that the steel is suited for the job. In other words the engineer takes responsibility, and puts his license and even freedom on the line, to insure that these external components are safe to use. The same principles hold in other licensed professions, for example, to nurses who must refuse to administer medications ordered by a doctor if the nurse believes the medication would be harmful to the patient.

Conclusions

The issue of licensure evokes strong feelings from software practitioners who do not believe that licensure should be required. But are these same individuals willing to risk a great deal on a software engineering decision? That is what a licensed professional must do – stake their reputations, treasure, livelihood, and even their freedom on the decisions they make. This level of risk tends to raise the standards of decision making, which is precisely the intent of licensure laws.

Of course, just as licensing does not prevent doctors from killing patience through malpractice, licensing software engineers will not prevent software errors from occurring, nor from preventing software based systems from harming the public. However, licensure does raise the standard of practice and provide assurance to the public of minimal competency on the part of practitioners.

It is not known how many software professionals will need to be licensed because the answer depends on how state legislatures choose to define systems that affect "health, safety and welfare." Each state will probably define these terms differently. But there is ongoing work in developing a "model law" for states to use [2].

References

[1] Texas State Law, § 1001.407, Construction of Certain Public Works, August 11, 2007, available at http://law.onecle.com/texas/occupations/1001.407.00.html

[2] P. Laplante and M. Thornton, "When do Software Systems Need to be Engineered," *Today's Engineer*, July 2011, available at http://www.todaysengineer.org/2011/Jul/licensure.asp

[3 Michael Davis. 2011. Will software engineering ever be engineering?. *Communications of the ACM* 54, 11 (November 2011), 32-34.

[4] Phillip A. Laplante, “Professional Licensing and the Social Transformation of Software Engineers,” *Technology and Society*, Summer 2005, pp. 40-45.

[5] Dennis Frailey, “Viewpoint: Licensing Software Engineers,” *Communications of the ACM,* vol. 42, no. 12, pp. 29–30, 1999.