Should Software Engineers Be Licensed?

John C. Knight
Department of Computer Science
University of Virginia

There is ample evidence that defective software is a causal factor in many failures of safety-critical systems in a variety of application areas. Some of these failures, such as the well-known Ariane V incident, are widely reported and carefully investigated, although appropriate corrective actions might not follow. Other failures, although serious, are far less known and tend not to be carefully studied. Software in automobiles, for example, has been the cause of numerous recalls, and the public has been exposed to serious hazards as a result. Yet the very existence of defective software systems in automobiles, their number, the significance of the defects, and the number of accidents that have occurred are not widely known; and there is no indication that the automobile industry has a comprehensive plan for dealing with the situation.

Why do software-related failures occur? In part—and, I would claim, in large part—it is because those who develop such systems are not adequately trained in the basic technology of software development. Although frequently they are able to produce software that provides certain basic functionality, they fail to understand or are completely unaware of topics such as the crucial importance of specification, the difficulties that arise in concurrent programs, the limitations of testing, the effects of rounding error, and the lack of timing predictability in modern processors.

Would the rate of failure be reduced to an acceptable level if some or all of the software developers who build safety-critical systems were licensed? The answer to this question depends on what one means by licensing, of course, but current proposals are little more than general examinations. The answer, therefore, is almost certainly "no" although most people think that licensing is either the solution or a large part of it.

In the discussions surrounding licensing, comparisons are often made to other branches of engineering. In the United States, engineers can be licensed in disciplines such as civil, electrical, and mechanical engineering. To be licensed, an individual has to: (1) complete a degree program in the relevant discipline from an accredited institution; (2) gain experience in the practice of engineering; (3) pass an extensive two-part technical examination; and (4) be recommended by a licensed engineer. Practicing as a licensed engineer also requires that the licensee maintain professional standards and work within a code of ethics. Professional societies are the primary means of enforcing engineering standards, and violation of the code of ethics by a licensed engineer can lead to disciplinary action by the associated professional society.

To enable meaningful and useful licensing in software engineering, i.e., achieving the same depth and breadth, would require an equivalent infrastructure. This becomes problematic immediately when one considers the situation in higher education. Most engineering disciplines are supported by degree programs in their field. Electrical engineers, for example, typically complete an electrical engineering degree. Those developers who have formal training in software engineering have usually obtained their training in a computer science or computer engineering degree program.

Certainly there are degree programs in software engineering, but a very small proportion of practicing software developers graduate from such programs. And by no means all who practice software engineering have formal training in the subject. Most have formal training in other fields.

In computer science and computer engineering programs, the training in software engineering is often limited and occurs primarily in a single course entitled "software engineering". The disparity here is very significant. The graduate from a typical computer science or computer engineering program can be counted on only to have received basic training in writing software in a high-level programming language. But knowing the syntax of Java does not make an individual a software engineer.

Courses labeled "software engineering" often attempt to show students basic elements of design for systems that are extremely simple and very small by practical standards. The design strategies are an attempt to codify some of what has become accepted practice by current practitioners, but often have little theoretical foundation and so are of questionable value, even to the student. In order to be a successful and responsible professional in the safety-critical field, a software developer must understand the intricacies of a large number of fields, including real-time systems, formal specification, dependability assessment, software safety and so on. In a typical computer science degree program, each of these topics might be an elective, and even the interested student would be unable to take more than one or two of the set.

Assessment by examination is problematic in software engineering also. In other branches of engineering, there are well-established scientific principles upon which the practice of engineering rests. These principles have evolved over long periods of time, are agreed upon by the community, and change relatively slowly. They have the desirable characteristic of enabling analysis of engineered systems and the establishment of important system properties. In structural analysis within civil engineering, for example, application of the finite-element method together with appropriate data on strength of materials permits the overall strength of the structure to be determined. It is clear, therefore, that a licensing examination in civil engineering has to cover this type of analysis. In software engineering, however, not only is there a substantial lack of analytic techniques, but there is not even an agreed-upon set of topics that the engineer in training should know. Thus, the creation of a licensing examination as part of the licensing process would require agreement on the subject matter relevant to the practice of software engineering in the safety-critical systems field. This will be difficult if it is possible at all, and it will certainly require protracted discussion.

Being unable to identify scientific principles leads to an inability to identify current professionals who use them. A part of licensing involves mentor training where an engineer applying for a license studies under a professional who has held a license for some time. In order to trust the mentoring process, it would be necessary to trust the mentor and his environment. With the current situation, this seems unlikely to be possible.

In practice, licensing is a very small part of the way that other disciplines achieve engineering quality. In those disciplines, a licensed engineer (but only one) has to approve the designs of systems to which the public will be exposed. In a typical development, thousands of engineers will contribute but only one has to be licensed. Thus, although licensing is available, it is not the case that practicing the discipline requires a licence. Clearly, for a large system, the one licensed engineer who approves the design could not possibly check all the details of the design—he has to trust all the other non-licensed engineers who contributed. Thus, as important as all of the explicit mechanisms used by other branches of engineering are, a major factor in the maintenance of quality is a notion of professionalism that pervades the actions of engineers. Note also that the approval of a design by a licensed engineer is only required for systems to which the public will be exposed. Other systems require no involvement by licensed individuals at all. Thus licensing is not the panacea that some appear to think it is. Many other changes will be needed in order to raise the level of software quality in safety-critical systems to an acceptable level.

Certainly mistakes in the development of software for safety-critical systems will continue to be made no matter how carefully the software is built, and failures will continue to occur—that is the way things are in engineering. But the current situation is unacceptable and far worse than most people realize, including people such as corporate managers and government regulators. The solution to the problem has four elements: (1) ensure that those building safety-critical software are properly trained; (2) ensure that they know how to apply the training that they have; (3) ensure that they understand the *limitations* of their training; and (4) just as in other fields where the consequences of failure are very high, ensure that practitioners are properly monitored by their colleagues, independent auditors and government regulators. The cumulative losses associated with software-related failures have become very high, and the situation must be addressed quickly and effectively.